Introduction to Evolutionary Computations

Akira Imada

Computer and Computer System Department
Brest State Technical University
Republic of Belarus

□ Contents:

(cont'd)

- Neuronal Darwinism

  'We have yet another evolution in our brain.' — Edelman.

  - ⋆ Fully-connected Neural Networks.
  - ⋆ Associative memory as a model of human memory.

    'We can enhance performance by pruning synapses.
    — But how?

- NP-hard Combinatorial Optimization Problem.
  - ⋆ Knapsack Problem.
  - ⋆ Traveler's Salesman Problem.
  - ⋆ Extensions to more general forms.
- Exploitation of Diploidy Chromosomes.
  - ⋆ Sorting Network Problem.
    - · What is the minimum number of comparisons?
    - · Does EC work better than human?
      — A competition: Human vs EC.
- Coevolution — Predator vs Prey.
- Evolutionary Game Theory.
  - ⋆ Prisoner's Dilemma.

(cont'd)

- A Visualization of High-D space — Summon Mapping by EC

  'EC Reveals a Geometry in High-dimensional Space.'

  ⋆ Sammon Mapping by EC.

- NN Revisited — Evolving its Architecture.
  — Can we evolve not only weights but also its architecture?

- Lamarckian Inheritance & Baldwin Effect
  — Not biologically plausible but ...

  ⋆ Lamarckian Inheritance.

  'Why giraffe's neck is so long?'

  ⋆ Baldwin Effect.

  'Would hard effort of one's own be inherited?'

- Search for Multiple Peaks Simultaneously

  ⋆ Multi-modal Optimization.

  'Can we search for multiple peaks at a time?'

  · Niching Method
    - Fitness sharing.
    - Crowding.

  · Speciation Method.

- Multi-objectives Optimization

  'What if we have multiple criteria that are trade-off?'

(cont'd)

- Variations of EC's
  - · GA, EP, ES and GP
- Commonly Used Test Functions — to learn more about EC's:
  - ⋆ Ten commonly used test functions.
  - ⋆ NK landscape: to control ruggedness of a fitness landscape.
  - ⋆ A problem that tries to cheat us: Royal Road Function.
  - ⋆ Hitch-hiker's Problem.
- There's no free lunch

  No Free Lunch Theorem.

  'There's no free lunch:
  — Be careful when we compare two methods.'
- Summary and Conclusions.
- Related Web-pages.

□ What are Evolutionary Computations?

— Then what for, and how?

(What?)

- Algorithms inspired by Biological Evolution, i.e.,

  'Survival of the Fittest.'

(What for?)

- To obtain (Near-optimum) solution(s) to
  - · not-solvable-analytically

    and/or
  - · very difficult problems

(But how?)

- All what we need to design are?
  - · How we represent the problem by chromosomes?
  - · How do we evaluate fitness?

□ How we evolve candidate solutions?

— From stupid solutions to an excellent one!

Determine how we represent candidate solutions (*phenotype*)
as a *chromosomes*[1] *(genotypes/individuals).*

$\Downarrow$

Create a *population* of random chromosomes
to construct the 1st *generation.*

$\Downarrow$

Evaluate *fitness* (how good they are?) of each of these individuals.

$\Downarrow$

Select two chromosomes according to the fitness such that
"the higher the fitness the more likely to be selected."

$\Downarrow$

Recombine the two chromosomes to produce
one child chromosome by *crossover* and *mutation.*

$\Downarrow$

Create the next generation by repeating selection & recombination.

$\Downarrow$

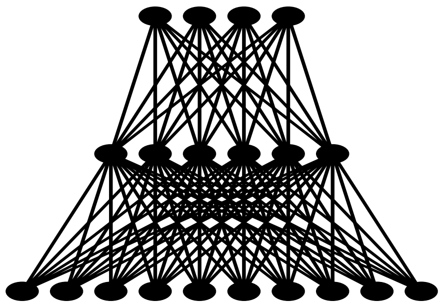
Thus, we expect better individuals from generation to generation.

[1] The name is in the sense that our chromosome give instruction or is interpreted on how to build a feasible phenotype like biological chromosome when phenotype is created.

□ A Simple Example

— Evolution of Weights of Feed-forward Neural Networks.

- NN as a black box.
  - ⋆ E.g. for Pattern Recognition.



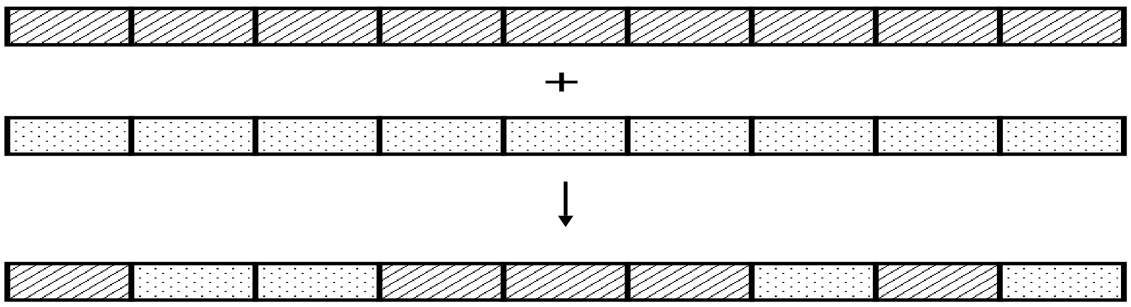
- But how we determine its weights?
  - ⋆ A proposal is by using ECs.

□ J_{ij} by Genetic Algorithm?

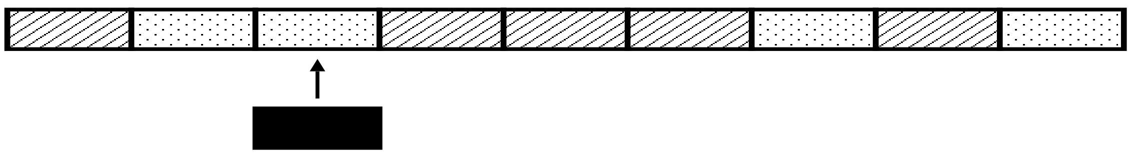
1. Represent a series of J_{ij} as a *population* of strings (*chromosome*).

J_{11}	J_{12}	J_{13}	···	J_{21}	·········	J_{N1}	···	J_{NN}

2. Define a fitness evaluation.
   (How good is each individual?)
3. Generate an initial *population* at random.
4. Evolve them with
   - *Selection*
   - *Crossover*



   - *Mutation* [2]



5. Better Solutions from *generation* to *generation.*

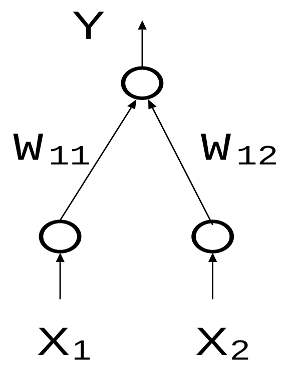
[2] A commonly used mutation rate is one over the length of chromosome.

□ Probably a Simplest examples.

- NN to solve AND & OR.

AND

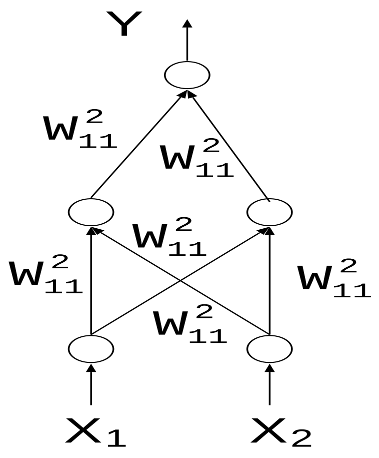
X_1	X_2	Y
0	0	0
0	1	0
1	0	0
1	1	1



· Well, how about NN to solve XOR?

XOR

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0



Exercise 1 *Obtain six weights values so that the above NN function as XOR.*

Exercise 2 *Create Pseudo code for EC to obtain the six weights above.*

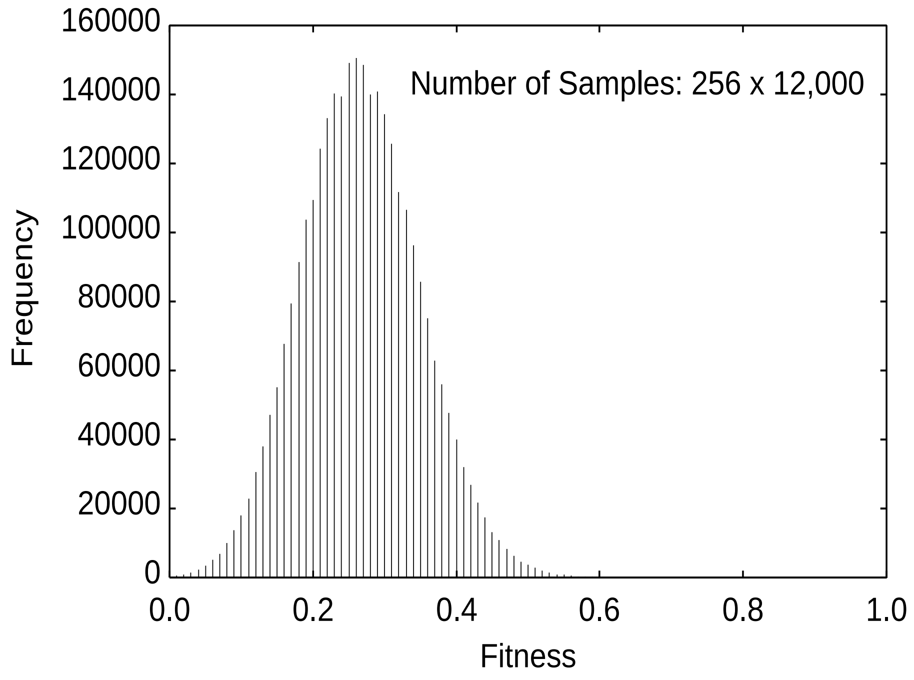
Pseudo Code for realization of NN to solve XOR by EC

```
step-1: Create an initial population of $N$ chromosomes with 6 genes each of whose values $\in [0,1]$ at random
        (Initialization).
step-2: Evaluate fitness of each of the $N$ chromosomes with the following procedures (Fitness Evaluation).
    (1) Set $i = 1$.
    (2) Pick up $i$-th chromosome from the population.
    (3) Create a NN (phenotype) from the chromosomes.
    (4) Give the NN a pair of inputs $x_1 = 0$ and $x_2 = 0$.
          · If $y = 0$ then $fitness(i) = fitness(i) + 0.25$
    (5) Give the NN a pair of inputs $x_1 = 0$ and $x_2 = 1$.
          · If $y = 1$ then $fitness(i) = fitness(i) + 0.25$
    (6) Give the NN a pair of inputs $x_1 = 1$ and $x_2 = 0$.
          · If $y = 1$ then $fitness(i) = fitness(i) + 0.25$
    (7) Give the NN a pair of inputs $x_1 = 1$ and $x_2 = 1$.
          · If $y = 0$ then $fitness(i) = fitness(i) + 0.25$
    (8) Set $i = i + 1$ and repeat from (2) to (8) until $i$ reaches $N$.
step-3: Sort $N$ chromosomes in the order of fitness value (with the higher one at the top.)
step-4: Select two chromosomes from the upper half of the population (Truncate Selection).
step-5: Create one child chromosome with the following two procedures (Reproduction).
  • Uniform cross-over
    (1) Set $j = 1$
    (2) Produce a binary random-number $x(j) \in \{0.1\}$
    (3) As the $j$-th gene of the child chromosome, if x(j) = 0 then pick up the $j$-th gene from the 1st
        parent otherwise from the 2nd parent.
    (4) Set $j = j + 1$ and repeat from (2) to (4) until $j = 6$.
  • Mutation
    (1) Set $j = 1$
    (2) Produce a continuous random-number $y(j) \in [0.1)$
    (3) As the $j$-th gene of the child chromosome, if $y(j) < 0.02$ then flip the gene value, i.e.,
          · if it is 1 replace it with 0 and if it is 0 vice versa.
    (4) Set $j = j + 1$ and repeat from (2) to (4) until $j = 6$.
step-6: Repeat from step-2 to step-6 until chromosome whose fitness is 1 appear.
```

□ What are ECs: Revisited.

— A little more in detail.

- To what degree are ECs efficient?
  - ⋆ An initial fitness distribution of some problem.



  - ⋆ *Needle-in-a-haystack* type of problems.

$$\Downarrow$$

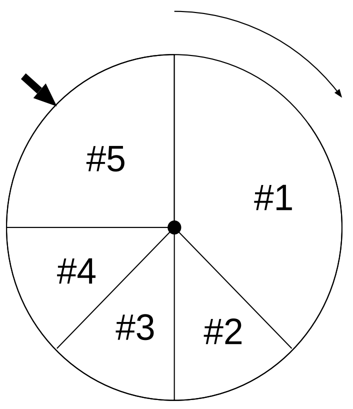
Almost everywhere low fitness except for the optimum.

  - ⋆ Random-Mutation-Hill-climbing vs EC.

(cont'd)

- Different versions of Selection.
  - ⋆ Truncation Selection
    - · Select parents from the best some-percentage of the population.
  - ⋆ Roulette Wheel Selection (Fitness Proportionate Selection)
    - · Select so that the probability to be selected is proportional to the fitness value.

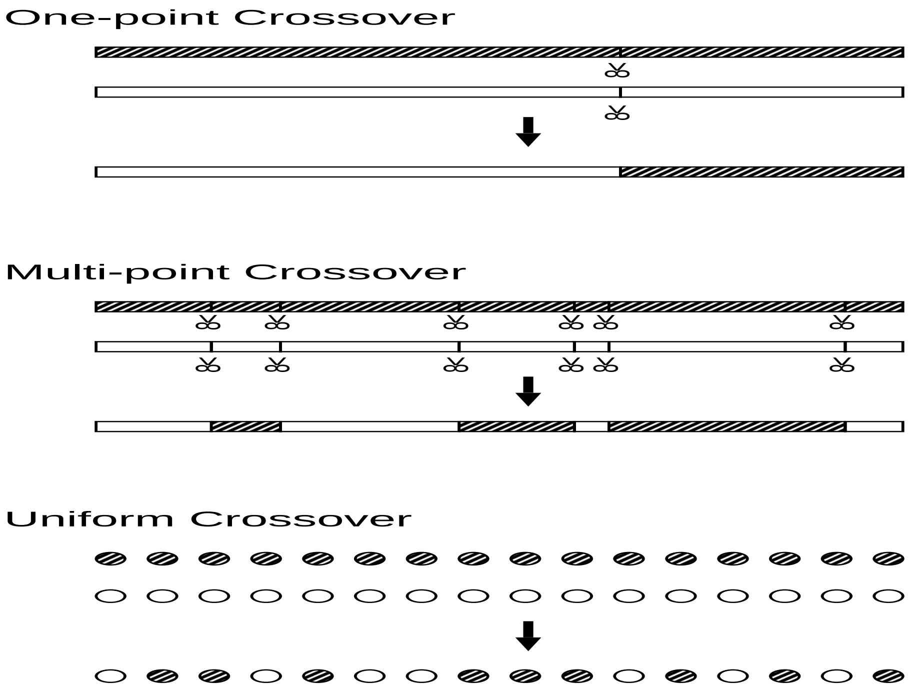
	fitness
individual #1	0.375
individual #2	0.125
individual #3	0.125
individual #4	0.125
individual #5	0.250



  - ⋆ Tournament Selection
    - · Assume we have the original μ parents and their μ children. The fitness value of each of the 2μ individuals are compared to those of q individuals which are chosen randomly from the whole 2μ points at every time of the comparison Then the 2μ points are ranked according to the number of wins, and the best μ points survive (q-tournament selection).

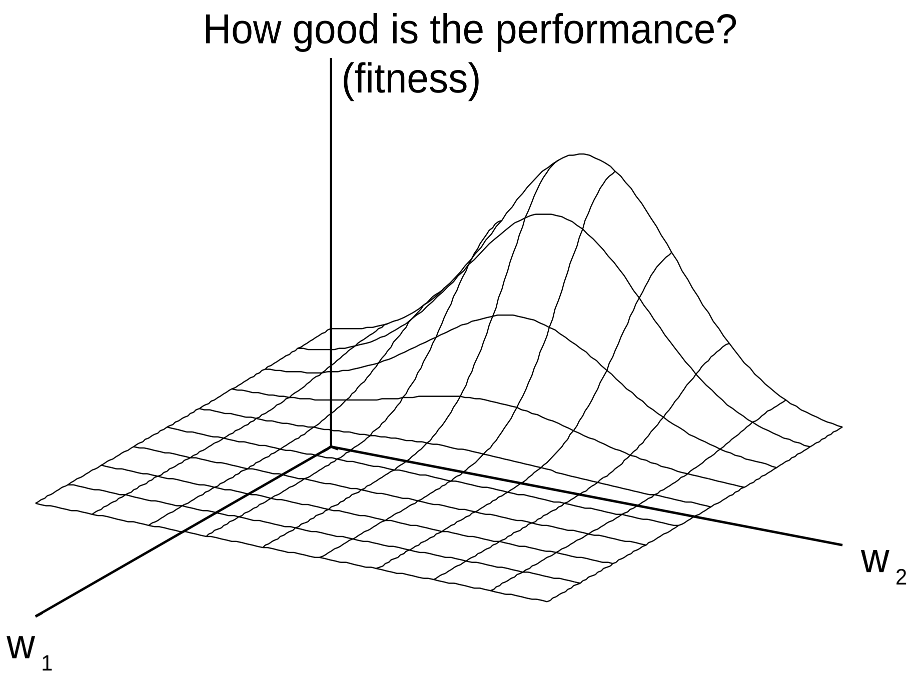
(cont'd)

- Different versions of Crossover.
  - ⋆ One-point Crossover.
  - ⋆ Multi-point Crossover.
  - ⋆ Uniform Crossover.



(cont'd)

- A Concept of Fitness Landscape.



□ Why do ECs work?

— Holland's original thought (formulated in 1975).

- *Building Block Hypothesis* & *Schema Theorem.*

⋆ What is schema?

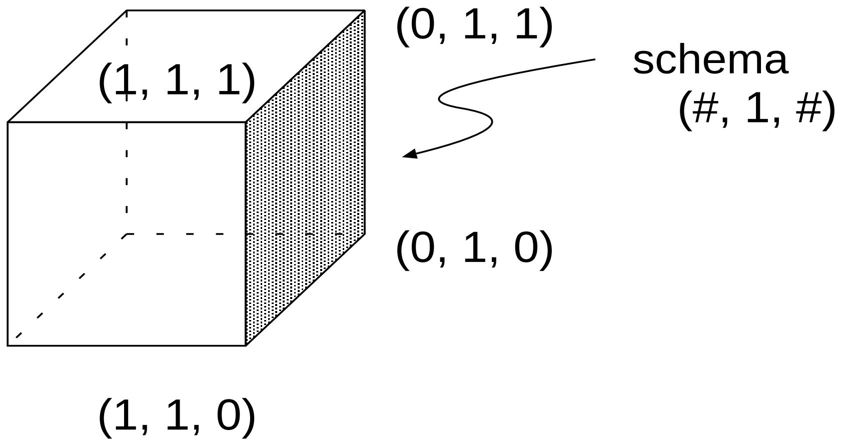
· A string of L element $\in \{1, 0, \#\}$ is called *schemata* (*schema* is the plural form of *schemata*),

· where # matches either 0 or 1 which we *don't care*, something like *wild-card*

- Schema of length L is a hyperplane of L-dimensional space,

$\Downarrow$

- while a binary string of length L is a vertex of L-dimensional hyper cube.



· Each binary string *instantiates* 2^L schemata.

(cont'd)

· Two parameters:

- *Order:*
  Number of 0 and 1 (no # symbols).

- *Defining Length:*
  Length from the first to the last defined position.

An example:

(1 0 0 # # # 0 # # #)
$\Rightarrow$ Order: 4
Defining-Length: 7

(1 0 0 1 0 1 0 1 0 1)
$\Rightarrow$ an *instance* of the schema above.

(cont'd)

⋆ Then what are *building blocks*?

· Building blocks are shorter pieces of an overall solutions.

$\Downarrow$

Combining small building blocks into larger building blocks.

· A metaphor of *features*.

$\Downarrow$

Combining features from two good parents expecting crossover to produce even better children.

So, *Building Block Hypothesis* expects

$\Downarrow$

Survival of good building blocks by iteratively combining to form larger better building blocks.

(cont'd)

Thus, *crossover* scatters the building blocks throughout the population.

$\Downarrow$

Sometimes, crossover recombines the worst features but if so, children will be less likely to survive.

$\Downarrow$

Sometimes, it will recombine the best features from two good individuals, creating even better.

$\Downarrow$

(cont'd)

$\Downarrow$

If two parents are instances of the same schema
the child will also be an instance.

$\Downarrow$

Thus, as the population converges, the search becomes more
and more focused on smaller and smaller subspaces of
the entire search space.

$\Downarrow$

That is,

if a particular schema gives high fitness values to its instances,
then the population is likely to converge on this schema

and once it so converges,
all offspring will be instances of this schema.

$\Downarrow$

(cont'd)

Assume two good features of *wildcat*: *sharp teeth* & *claws.*

$\Downarrow$

Also assume *sharp teeth* is specified by the genes "1" on the 2nd and "0" on the 3rd locus.

$\Downarrow$

So, schema bellow has a high fitness value.
{# 1 0 # # # # # # #}

$\Downarrow$

On the other hand, *sharp claws* are specified by the genes on the 1st, "0" on the 5th, and "1" on the 6th.

$\Downarrow$

So, schema bellow has also a high fitness value.
{1 # # # 0 1 # # # #}

$\Downarrow$

Thus in the next generation, schema with higher order
{1 1 0 # 0 1 # # # #}
will prevail. and some children are expected to have both features.

(cont'd)

To create a new individual that combines the best feature from each of parents.

$$\Downarrow$$

starting with randomly recombining features scattered throughout the population;

$$\Downarrow$$

expecting to combine small building blocks into larger building blocks.

$||$

Building Block Hypothesis

(cont'd)

Theorem 1 (Schema Theorem) *In Genetic Algorithms under Roulette Wheel Selection,*

$$n(H,t+1) \geq n(H,t) \cdot \frac{\hat{u}(H,t)}{\bar{f}(t))} \cdot (1 - p_{\text{disrupt}}(H,t))$$

where

· $n(H,t)$ *is expected number of instances of hyperplane* H *in the population at time* t*;*

· $\hat{u}(H,t)$ *is average fitness of the instances of* H *at time* t*;*

· $\hat{\bar{f}}(t)$ *is average fitness of individuals in the population;*

· $p_{\text{disrupt}}(H,t))$ *is the probability of disruption due to genetic operator like crossover and mutation.* [3]

$$\Downarrow$$

The number of short-definition-length hyper-planes
with above average fitness is expected to grow rapidly,
while
those with below average generally decline.

[3] The probability is very small for H with short defining length, for example, i.e., Short defining length have a small chance of disruption by one point crossover and mutation.

(cont'd)

- Proof of the schema theorem
  - ⋆ Let H be a schema with at least one instance present in the population at time t. (When $x \in H$, we say x is an instance of H.)
  - ⋆ Let $m(H,t)$ be the number of instances of H at time t and let $\hat{u}(H,t)$ be the observed average fitness of H (averaged fitness of all instances of H in the population) at time t.
  - ⋆ What we want is $E(m(H,t+1))$, the expected number of instances of H at time $t+1$.
  - ⋆ Assume the selection is fitness proportionate, i.e., the expected number of offspring of x is equal to $f(x)/\bar{f}(t)$ where $f(x)$ is the fitness of x and $\bar{f}(t)$ is the average fitness of all individuals in the population at time t.
  - ⋆ By definition

$$\hat{u}(H,t) = \sum_{x\in H} f(x)/m(H,t)$$

hence

$$\begin{aligned} E(m(H,t+1)) &= \sum_{x\in H} f(x)/\bar{f}(t) \\ &= (\hat{u}(H,t)/\bar{f}(t))m(H,t). \end{aligned}$$

(cont'd)

- ⋆ Thus even though the GA does not calculate $\hat{u}(H,t)$ explicitly, the increases or decreases of instances of schemas in the population depend on this quantity.
- ⋆ Crossover and mutation can both destroy and create instances of H. But for now let us include only the destructive effects — those that decreases the number of instances of H.
- ⋆ Let p_c be the probability that single-point crossover going to be applied to a string, and
- ⋆ an instance x of schema H be picked as a parent.
- ⋆ Then schema H is said to survive under single point crossover if one of the offspring is still an instance of schema H.
- ⋆ A crossover that occurs within the defining length of H can destroy H.

(cont'd)

- ⋆ So, the fraction of x that H occupies multiplied by p_c:

$$p_c \cdot (d(H)/(l-1))$$

gives us an upper bound on the probability with which it will be destroyed.

(The value is an upper bound because some crossovers inside a schema's defined positions will not destroy it.)

- ⋆ Subtracting this value from 1 gives a lower bound on the probability of survival $S_c(H)$,

- ⋆ Thus we can give a lower bound on the probability $S_c(H)$ with which H will survive single-point crossover as

$$S_c(H) \geq 1 - p_c(d(H)/(l-1))$$

where $d(H)$ is the defining length of H and l is the length of x.

- ⋆ In short, the probability of survival under crossover is higher for shorter schemas.

(cont'd)

⋆ Then the disruptive effects of mutation. Let p_m be the probability of any bit being mutated.

⋆ Then $S_m(H)$, the probability with which schema H will survive under mutation of an instance of H, is equal to

$$(1 - p_m)^{o(H)}$$

where $o(H)$ is the order of H.

⋆ That is, for each bit, as the probability with which the bit will not be mutated is $1 - p_m$, so the probability with which no defined bits of schema H will be mutated is this quantity multiplied by itself $o(H)$ times.

⋆ In short, the probability of survival under mutation is higher for lower-order schemas. Therefore

$$E(m(H, t+1)) \geq$$

$$\frac{\hat{u}(H,t)}{\bar{f}(t)} \cdot m(H,t) \cdot (1 - p_c \cdot \frac{d(h)}{l-1}) \cdot (1 - p_m)^{o(H)}$$

⋆ This is the perfect version of the schema theorem.

(cont'd)

- Interpretation again.
  - ⋆ The short low-order schemas whose average fitness remains above the mean will receive exponentially increasing numbers of samples over time, since the number of samples of those schemas that are not disrupted and remain above average in fitness increases by a factor of $\hat{u}/\bar{f}$ at each generation.
  - ⋆ In evaluating a population of n strings, the GA is implicitly estimating the average fitnesses of all schemas that are present in the population, and increasing or decreasing their representation according to the schema theorem.
  - ⋆ This simultaneous implicit evaluation is known as implicit parallelism.

□ Yet Another Evolution of Neural Networks.

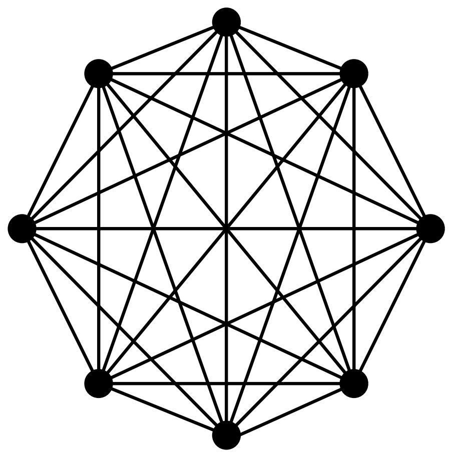
- Underlying Idea:

Edelman's Evolution of Neuron's level

vs.

Darwinian Evolution of Species' Level.

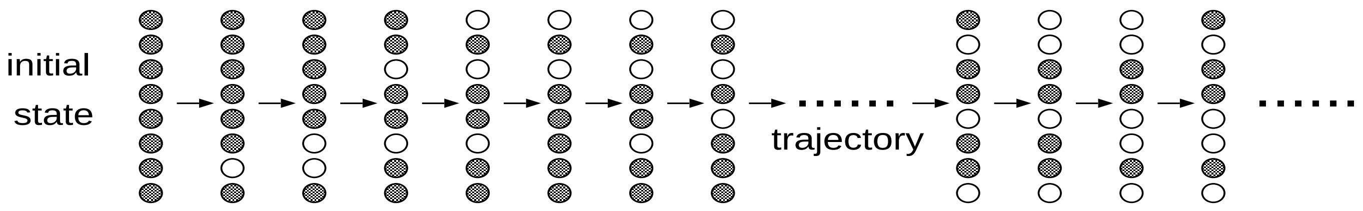
- Fully-connected Neural Networks (as a Dynamic System):



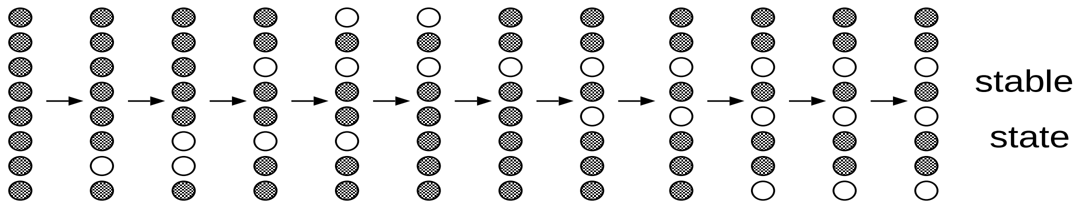
(cont'd)

- Associative Memory as a Dynamic System.

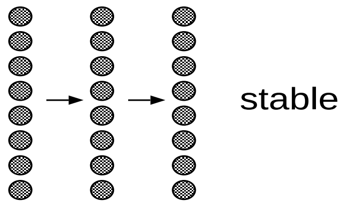
Chaotic Trajectory:



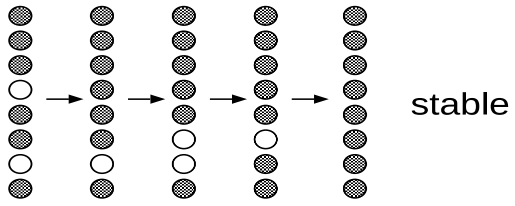
Convergence to a Spingras:



A Fixed Point Attractor:



Recovery from Noise:



(cont'd)

- Associative Memory

  — as a Model of Human Memory

  – Associative memory
    - ∗ stores patterns
      - · in a distributed way (among neurons),
    - ∗ recalls patterns
      - · from noisy and/or partial input.

  $\Downarrow$ i.e.

    - ∗ gives us
      - · perfect recollection from imperfect information

    - ∗ has been realized mainly with
      - · Fully-connected Neural Network Model;

      though it could be also realized with other model like.
      - · Artificial Immune System Model;
      - · spiking Neurons:

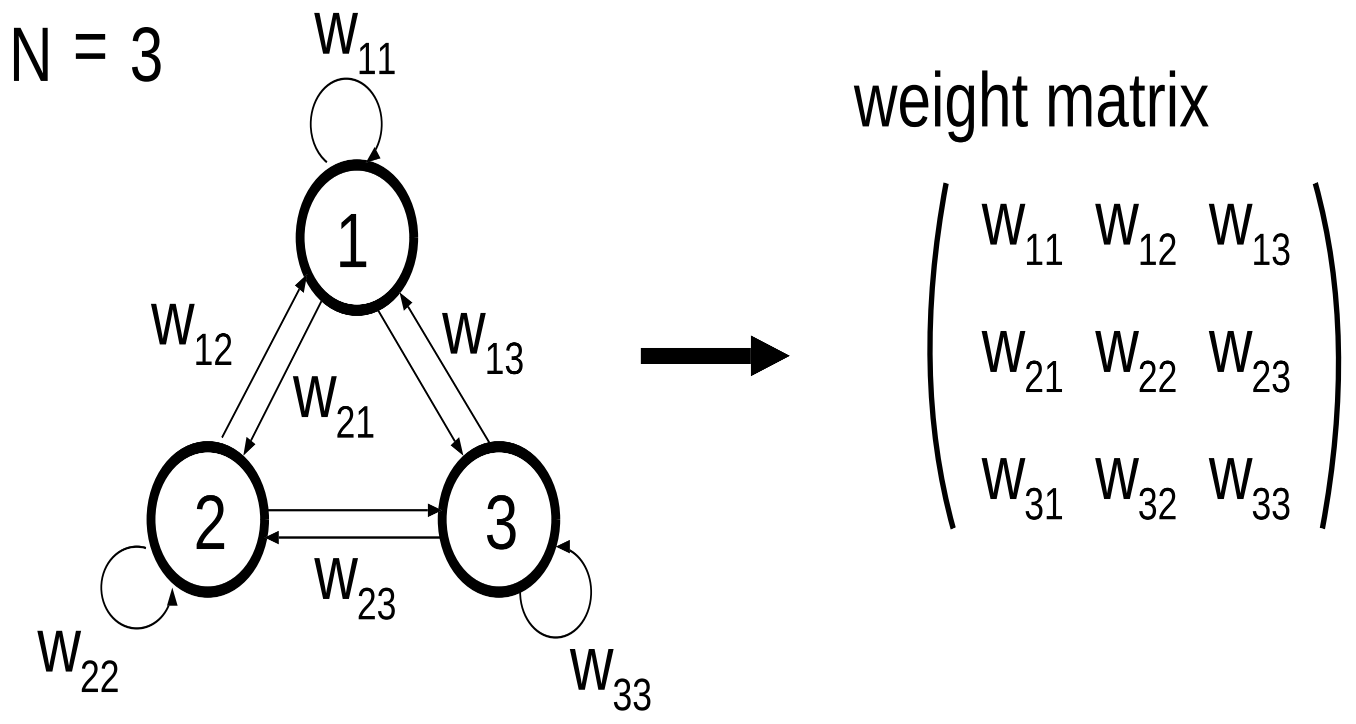
(cont'd)

- State Transition of the Hopfield Model

$$s_i(t+1) = sgn(\sum_j^N w_{ij} s_j(t))$$

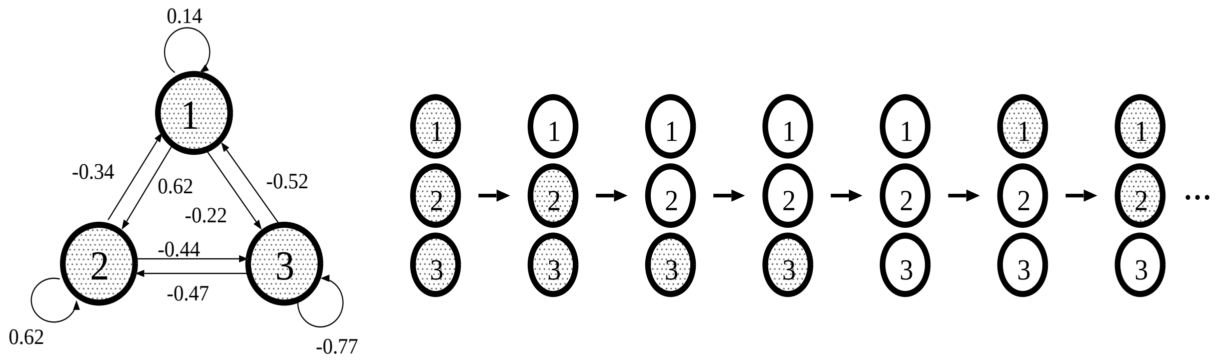
$\Downarrow$ e.g.

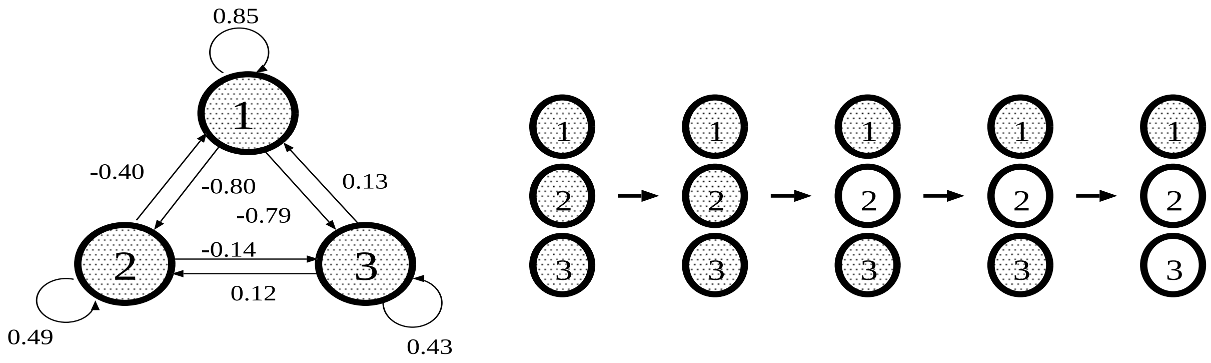
$$\begin{aligned} s_1(1) &= sgn\big(w_{11} \cdot s_1(0) + w_{12} \cdot s_2(0) + w_{13} \cdot s_3(0)\big) \\ &= sgn\big(0.14 \cdot 1 + (-0.34) \cdot 1 + (-0.52) \cdot 1\big) \\ &= -1 \end{aligned}$$

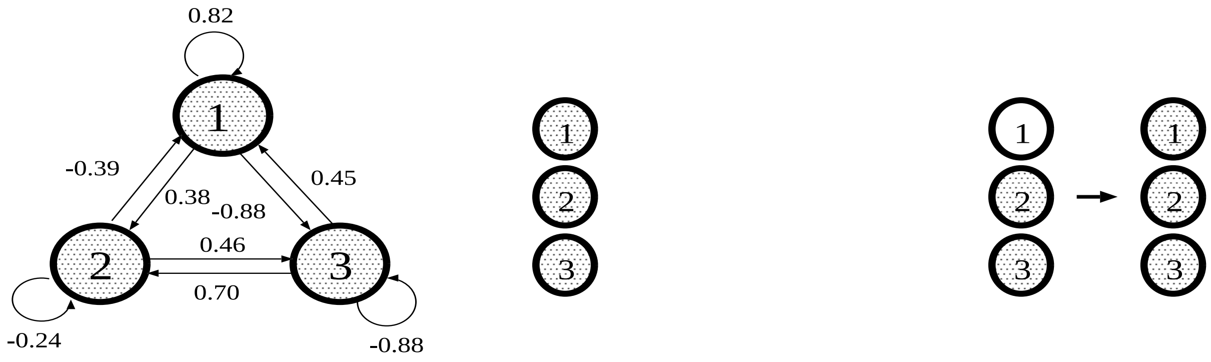


(cont'd)

- Toy Examples







(cont'd)

- Hebbian Weights

$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^{p} \xi_i^{\nu} \xi_j^{\nu}, \quad (i \neq j), \quad w_{ij} = 0.$$

To store (1, 1, 1) in our previous toy example:

$$\begin{pmatrix} 0 & 0.11 & 0.11 \\ 0.11 & 0 & 0.11 \\ 0.11 & 0.11 & 0 \end{pmatrix}$$

$$\Uparrow$$

Note: this is a different solution than the one in the previous slide

- Storage enhancement by pruning synapses.

□ Combinatorial Optimization Problem

— EC applications to *NP complete problems*[4] .

- 0-1 Knapsack Problem

  ⋆ When n items whose i-th item has wight w_i and profit p_i, search for a binary vector

  $$\mathbf{x} = (x_1, x_2, \cdots, x_n)$$

  that maximizes

  $$\sum_{i=1}^{n} x_i p_i.$$

  such that

  $$\sum_{i=1}^{n} x_i w_i \leq C$$

  where C is capacity of the knapsack.

  ⋆ Assume $\mathbf{x}$ be chromosome of our EC.

  ⋆ We sort all items in decreasing order of p_i/w_i then we take i-th item iff i-th gene of chromosome $\mathbf{x}$ is 1 until knapsack is full or there are no items left.

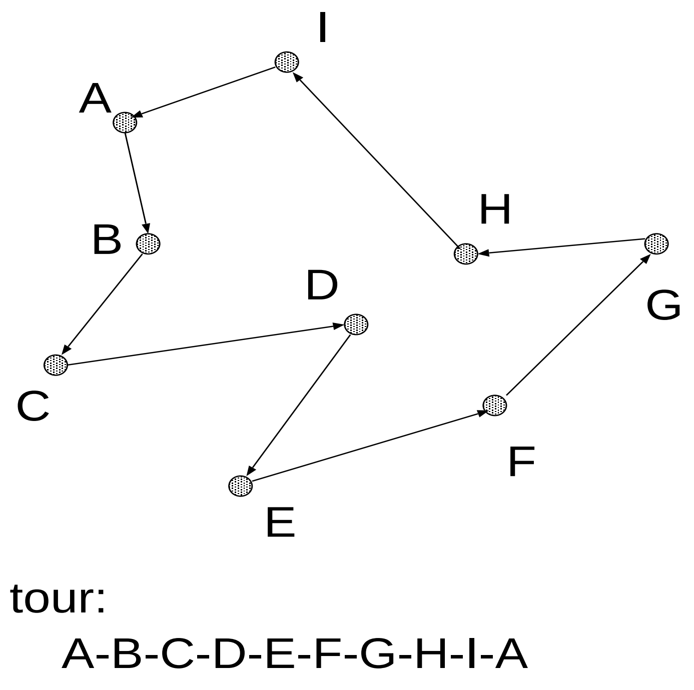
[4] If problems have a nondeterministic polynomial-time solution but have no known polynomial-time solution the problem is called NP complete

(cont'd)

- Traveling Salesman Problem
  - ⋆ A salesman must visit a number of cities, and then return home. In which order should the cities be visited to minimize the distance traveled?
  - ⋆ If we represent a candidate solution with a list of cities to be visited in order, the results of crossover and mutation are feasible?
  - ⋆ In order for the result of crossover and mutation to be feasible what representations are possible?

(cont'd)

An example:



Chromosome is to be designed so that it allows us to form a tour by following the procedure:

Step-1. Set $i = 1$.
Step-2. If i-th gene is n then n-th city in the list is the city to be currently visited.
Step-3. Remove the city from the list.
Step-4. Set $i = i+1$ and repeat Step-2 to Step-4 while $i \leq n$.

(cont'd)

For example, when the list of cities is

$$\{\text{A, B, C, D, E, F, G, H, I}\}$$

chromosome: (112141311) is the tour:

A-B-D-C-H-E-I-F-G.

Exercise 3 *Try some one-point crossover on two parents* (112141311) *and* (515553321)

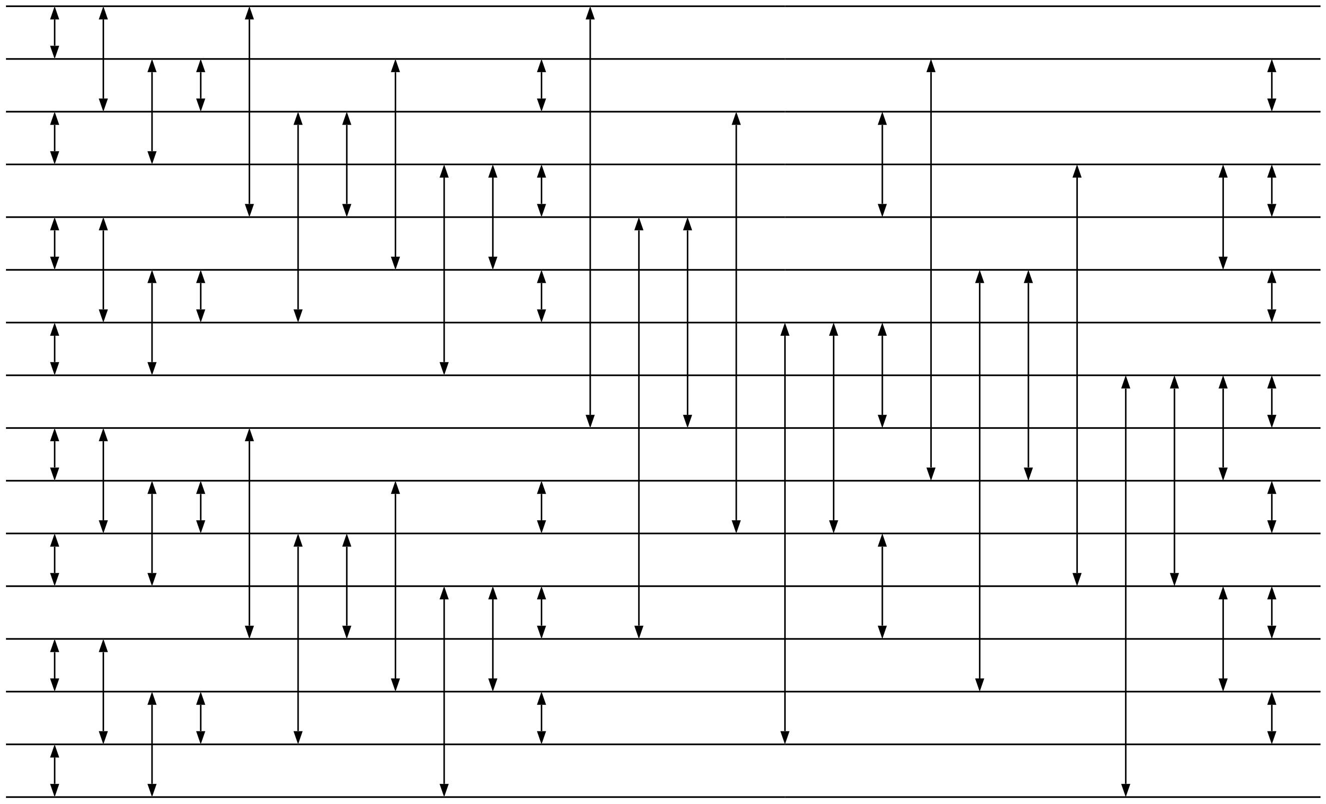
⋆ Then how we design mutation?

- How about specifying two points at random and reverse the gene order?

□ Exploitation of Diploidy Chromosomes.

- Sorting Network Problem.
  - ⋆ The task is to sort n items,
    - · For the purpose, the i-th and j-th element are compared and swap if necessary.

Batcher Sort: 63 comparisons by Knuth (1973)



Comparisons in the same column can be made in parallel.

- ⋆ What is the minimum number of comparisons?

(cont'd)

⋆ Competition among human researchers. ($n = 16$)

- 65 comparisons Bose and Nelson (1962).
- 63 by Batcher and by Floyd and Knuth (1964).
- 62 by Shapiro (1969)
- 60 by Green (1969)

$$\Downarrow$$

But still no proof for optimality.

- Then does EC work better than by human?

$$\Downarrow$$

- Hillis's challenge using diploid chromosome.

(cont'd)

⋆ **Hillis's two Innovations:**

1. Diploidy Chromosome.

   - Each individual consists of 15 pairs of 32-bit chromosomes.

(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)
(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)
(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)
(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)
(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)
(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)
(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)
(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)
(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)
(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)	(1001 1000 1010 1101 1110 0100 1110 0011)

   - Each chromosome consists of eight 4-bit string (called *codons*).

     (0001 0010 0101 1000 0000 0100 1111 1001)
     (0011 0100 0101 1000 1101 1100 1111 1001)

   - Each codon represents an integer between 0 and 15 indicating which item is to be compared out of 16.

     (01 02 05 08 00 04 15 09)
     (03 04 05 08 13 12 15 09)

   in the above example.

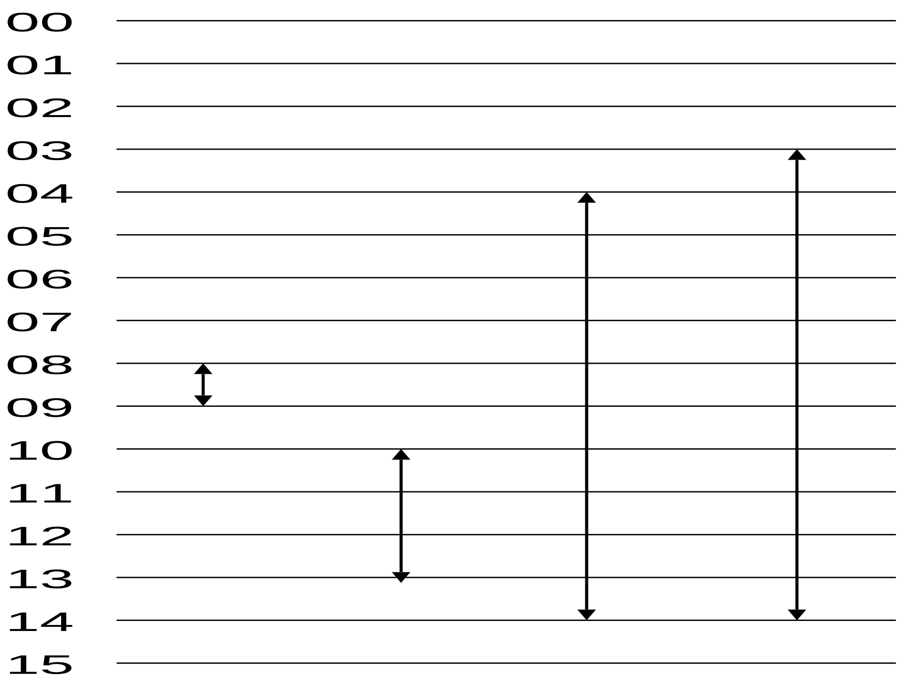
(cont'd)

- Each adjacent pair of codons in a chromosome specifies a comparison between two elements. Thus each chromosome encodes four comparisons.

- For example

(09 08 10 13 14 04 14 03)

indicates the four comparisons bellow.



- The chromosome pairs is read off from left to right.

(cont'd)

- If these adjacent two codons are same at the same corrisponding two positions of the chromosome pair (called *homozygous*) then only one pair of numbers is inserted in the phenotype. If it encodes different pairs of numbers (*heterozygous*) then both pairs are inserted in the phenotype.

- So in the previous example:

  (01 02 05 08 00 04 15 09)
  (03 04 05 08 13 12 15 09)

  means the following six comparisons:

  01⇔02, 03⇔04, 05⇔08, 00⇔04, 13⇔12, 15⇔09

- Thus 15 pairs of chromosomes produce a phenotype with 60-120 comparisons. The more homozygous positions, the fewer comparisons.

(cont'd)

- When two individuals are selected, one-point-crossover takes place within each chromosome pair inside each individual.
- For each of the 15 chromosome pairs, a crossover point is chosen at random and a single chromosome (called *gamete*) is formed.
- Thus 15 gametes from each parent are created.
- Each of the 15 gametes from the first parent is then paired with the gamete of the corrisponding position from the second parent to form one offspring.

□ Coevolution

In a usual sense

⋆ *Flowering Plants* vs *Pollinating insects.*

But what is discussed here is

⋆ *Predator* vs *Prey.*

$\Downarrow$

success on one side = failure on the other.

---

evolutionary pressure
for prey to defend themselves better
(to run faster, good eyesight, etc.)

while

for predators to develop attacking strategies.
(strong claws, faster flying, etc.)

$\Downarrow$

Arms race

(cont'd)

Hillis' two populations:

$\Downarrow$

Sorting networks & sets of test lists of 16 numbers

$\Downarrow$

Fitness for a sorting network:
percentage of correctly sorted test lists.

vs

Fitness of the set of lists to be sorted:
percentage of test lists incorrectly sorted by the network.

(cont'd)

Two different populations: {*Candidate Solutions*} *&* {*Test*}

$\Downarrow$

each solution is paired with a randomly chosen test N times.

$\Downarrow$

if a solution solves a test correctly, payoff will be 1 otherwise 0

$\Downarrow$

$$\{\textit{fitness of the solution}\} = \frac{\sum_{k=1}^{N} \{\textit{payoff}\}_k}{N}$$

$\Downarrow$

$\star$ *each test is paired with N randomly chosen solutions.*

$\Downarrow$

Fitness in a similar way but inversed one.

$\Downarrow$

Solutions that solve more tests will be fitter, and fitter tests are more often violated by more solutions.

□ Yet Another Example: Evolution of Strategy.

- Prisoner's Dilemma.[5]
  - ⋆ Two arrested prisoner A and B are offered a deal:
    - · If A confesses and B does not, A will be forbidden and B will get 5 years in jail, and vice versa.
    - · If both confess, then both will get 4 years in jail.
    - · If both do not they will each get 2 years.
  - ⋆ So the reward that A/B will receive is summarized as:

$B\backslash A$	Cooperate	Defect
Cooperate	3/3	0/5
Defect	5/0	1/1

$$\Downarrow$$

D. increases one's own reward at the expense of the opponent while *C.* increases the reward for both players, fewer though.

$$\Downarrow$$

In any case it would be better to defect!

[5] Proposed by Merrill Flood and Melvin Dresher in the 1950's

(cont'd)

⋆ In more general

$B \backslash A$	Cooperate	Defect
Cooperate	γ_1/γ_1	γ_2/γ_3
Defect	γ_3/γ_2	γ_4/γ_4

On the condition to be *Dilemma.* [6]

$$2\gamma_1 > \gamma_2 + \gamma_3$$

$$\gamma_3 > \gamma_1 > \gamma_4 > \gamma_2$$

[6] Papopurt (1966)

(cont'd)

But how about if the game is to be repeated?

$\Downarrow$

Strategy: What would be the optimum next action?

$\Downarrow$

$\star$ Always Defect
or
$\star$Tit-for-Tat:
Cooperate on the first play, and afterward
the same action as the opponent in the previous game.

$\Downarrow$

Here, strategy determines the next action based on three previous moves in a raw.

$\Downarrow$

Number of all possible previous three games is $2^8 = 64$
64 combination of Cooperate and Defect.

□ A Visualization of High-D space
— Summon Mapping[7] by EC

- Assume N points are given in the n-D space.

- Calculate distance matrix R $(N \times N)$ whose i-j element is the Euclidean distance between the i-th and j-th point.

- Also think of a tentative N points in the 2-D space that are located at random at the beginning.

- The distance matrix Q is calculated in the same way as R.

- Then the error matrix $P = R - Q$ is defined.

- Search for the locations of N points in the 2-D space that minimizes the sum of element P.

This is an optimization problem which we can solved by EC,

- N points in n-D space $\Rightarrow$ N points in 2-D space

  with the distance relation being preserved as much as possible; or

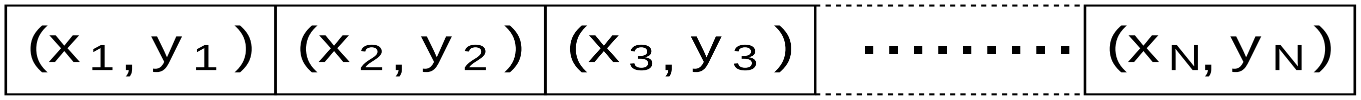
- The n-D distances are approximated by 2-D distances

  with a minimal error.

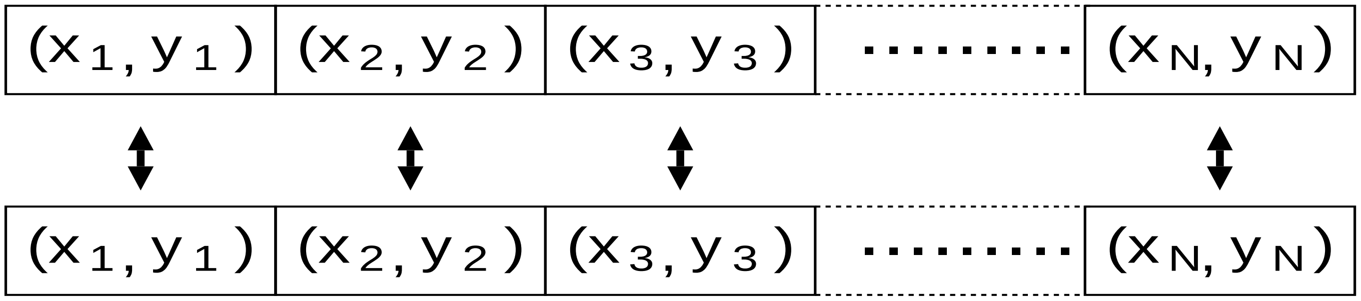
[7] by J. W. Sammon (1969).

□ GA Implementation of Sammon Mapping

Chromosome:

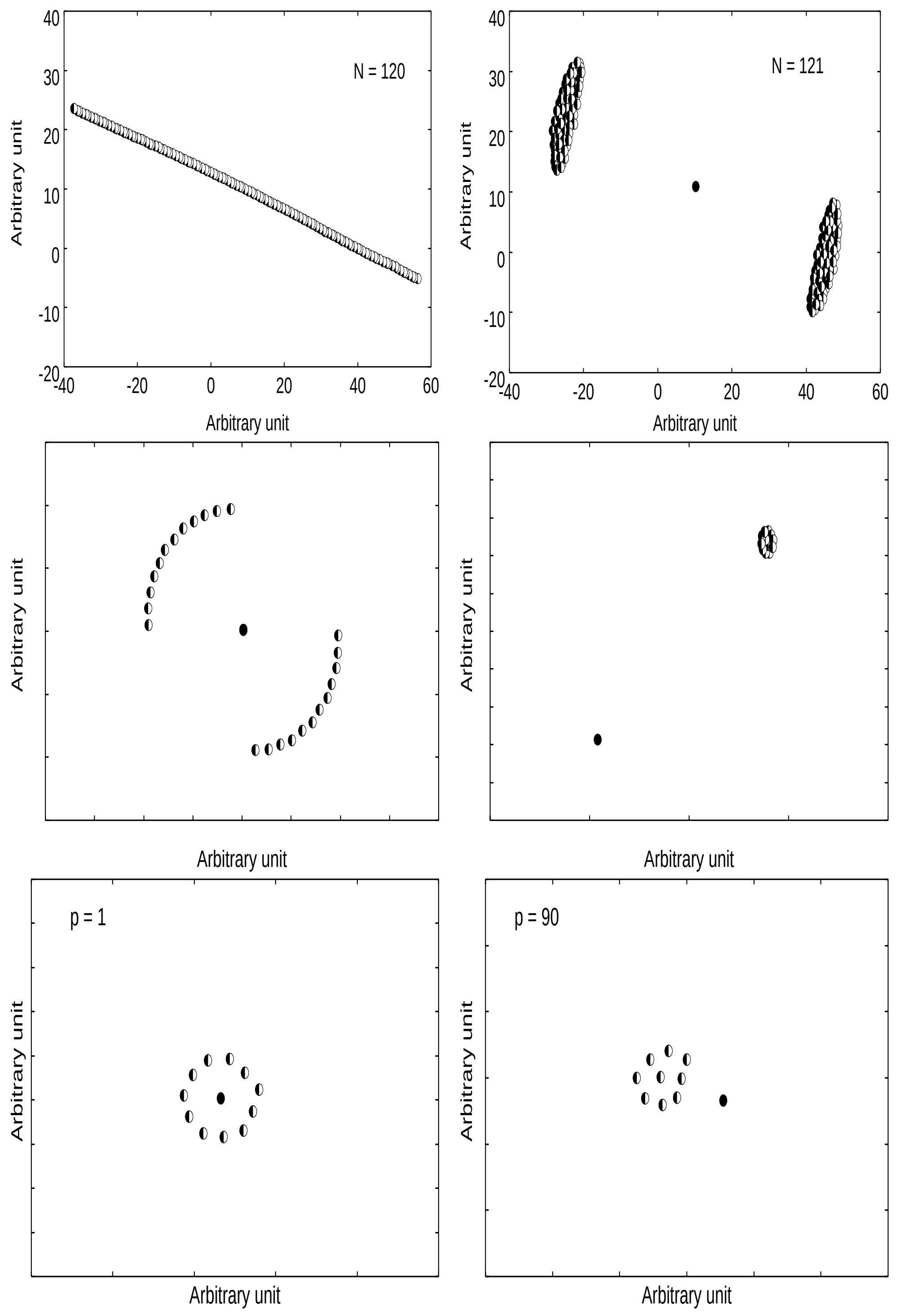


Recombination with Uniform Crossover:

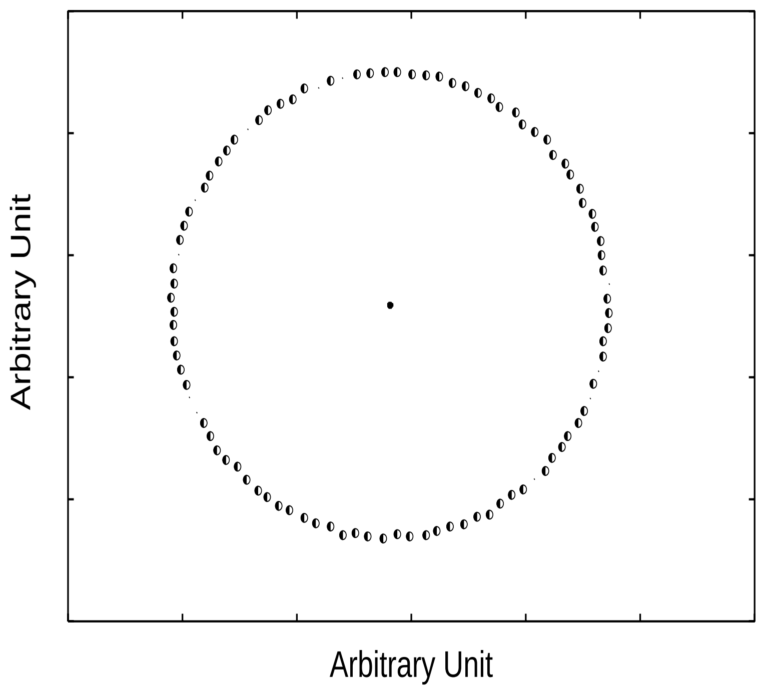


Mutation:

□ Examples in $49^2 = 2401$ dimensional space:



□ By adding points near the Origin:



□ NN Revisited — Evolving its Architecture

Evolving only weights on a fixed topology
vs.
Evolving topologies along with weights.

But different topology has a different length of chromosome.

$\Downarrow$

How we crossover them?

Let's study here NEAT [8]
(Neuro Evolution of Augmenting Topology)
out of many so-far-proposals.

[8] by Stanley & Mikkulainen (2001).

(cont'd)

- Start with a minimal structure with no hidden layer (instead of random topologies).
  - ⋆ This is for the simplicity for its implementation, but naturally appealing too, since complexity in nature develops from simple to complex.

- Gene is 5-tuple.

  (1) historical origin of the gene
  (a chronology of the appearance of every gene.)
  - · At the beginning, every individuals has a identical length, and 1, 2, $\cdots$ are assigned from the top gene to the end gene.
  - · Whenever a new gene appears (e.g. by mutation), the number is incremented and assigned to the gene.
  - · The number will never change thereafter.

  (2) input node

  (3) output node

  (4) weight value
  - · weight values of the connection between input and output node.

  (5) enable or disable

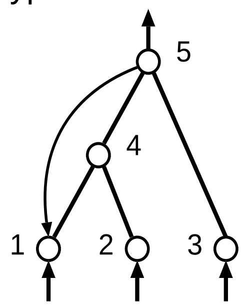
(cont'd)

- An example of its genotype & phenotype

Genotype

Chronogical order 1 In 1 Out 4 Weight 0.7 Enable	Chronogical order 3 In 2 Out 4 Weight -0.5 Enable	Chronogical order 4 In 2 Out 5 Weight 0.5 Disable	Chronogical order 6 In 4 Out 5 Weight 0.4 Enable	Chronogical order 9 In 5 Out 1 Weight -0.6 Enable

Phenotype

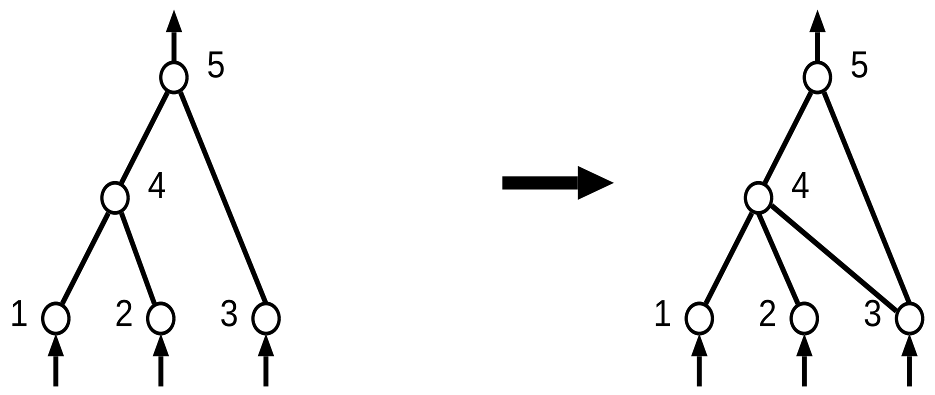


(cont'd)

- Two forms of Mutation
  - ⋆ Add-connection-mutation
    - · A single new-gene is added to the end of chromosome connecting two previously unconnected nodes.
    - · and the next available chronological gene number is given.

#1 1 -> 4	#3 2 -> 4	#4 2 -> 5 Dis	#5 3 -> 5	#6 4 -> 5

#1 1 -> 4	#3 2 -> 4	#4 2 -> 5 Dis	#5 3 -> 5	#6 4 -> 5	#7 3 -> 4



(cont'd)

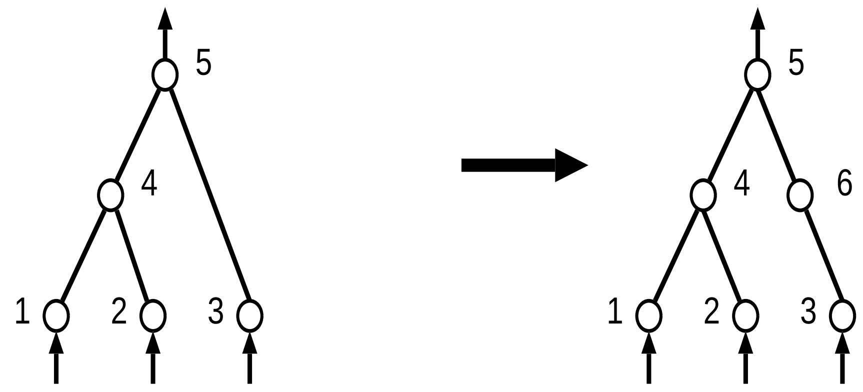
⋆ Add-node-mutation

· One gene is disabled and two new genes a re added to the end, that is,

· An existing connection is split and the new node placed where the old connection used to be. Old connections *disabled*

#1 1 -> 4	#3 2 -> 4	#4 2 -> 5 Dis	#5 3 -> 5	#6 4 -> 5

#1 1 -> 4	#3 2 -> 4	#4 2 -> 5 Dis	#5 3 -> 5 Dis	#6 4 -> 5	#8 3 -> 6	#9 6 -> 5



· Two new connections are added to the chromosome.

· Connection to the new node is assigned weight 1.

· Connection from the new node remains the same as the old one.

⋆ Both expand the size of chromosome.

(cont'd)

- Crossover

Two genes with the same historical origin are both derived from the same ancestral gene of a moment in the history.

$\Downarrow$

So, we now know which genes match up with which gene

$\Downarrow$

Genes that do not match are either
disjoint (not match in the middle of chromosome) or
excess (not match in the end.)

$\Downarrow$

Matching genes are inherited at random, and
disjoint and excess ones are inherited from the fitter parent.

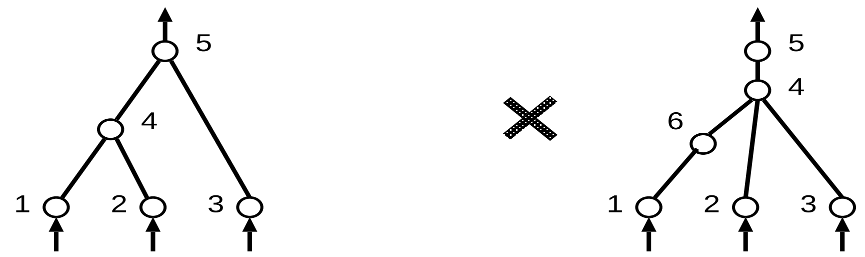
(cont'd)

parent 1

#1	#2	#4	#5	#6
1 -> 4	2 -> 4	2 -> 5	3 -> 5	4 -> 5
		Dis		

parent 2

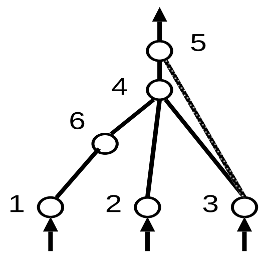
#1	#2	#3	#4	#6	#7	#8
1 -> 4	2 -> 4	3 -> 4	2 -> 5	4 -> 5	1 -> 6	6 -> 4
Dis			Dis			



#1	#2		#4	#5	#6		
1 -> 4	2 -> 4		2 -> 5	3 -> 5	4 -> 5		
			Dis				
#1	#2	#3	#4		#6	#7	#8
1 -> 4	2 -> 4	3 -> 4	2 -> 5		4 -> 5	1 -> 6	6 -> 4
Dis			Dis				

offspring

#1	#2	#3	#4	#5	#6	#7	#8
1 -> 4	2 -> 4	3 -> 4	2 -> 5	3 -> 5	4 -> 5	1 -> 6	6 -> 4
Dis			Dis				



□ Not Biologically Plausible but...

— An effect of life time learning on evolution.

- Baldwin Effect.

  ⋆ Results of learning of individuals during their life time do not change their chromosomes, but only affect the selection after fitness evaluation.

  ⋆ The model of Hinton & Nowlan (1987).

  - Assume chromosomes of 20 genes of 0's and 1's.
  - Only one configuration out of 2^{20} is assigned fitness one with others being zero.
  - Like a *hole in one!*

$\Downarrow$ learning at life time:

  - Hinton & Nowlan's experiment.

$$\text{Set}\left\{\begin{array}{c} 0 : 25\% \\ 1 : 25\% \\ ? : 50\% \end{array}\right\}\text{of the genes on overage.}$$

  · 1000 learning trials with each of which assigning "?" position either 0 or 1 at random.

  · Then fitness is evaluated as

$$f = 1 + 19(999 - n)/1000$$

with the highest possible value being 20 while the lowest 1.

(cont'd)

- Lamarckian Inheritance.
  - ⋆ Lamarck believed acquired characteristics during life-time could be passed to its offspring;

    'Would hard effort during one's life time be inherited?' $\Rightarrow$ The answer is 'No!' but,
  - ⋆ Incorporation of learning results into chromosomes sometimes enhances the performance of GAs

□ Variations of EC's:

— GA, EP, ES and GP

- Evolutionary Programming (EP)

  ⋆ Evolution with only mutation (no crossover).

  - Assume μ points in n-dimensional space construct a population.

  - Each point has additional n variables σ_i in addition to its coordinates x_i.

  - In each generation, each of μ points is mutated by

    $$x_i^{\text{new}} = x^{\text{old}} + \sigma_i$$

    which produces μ mutants.

  - After this mutation, these σ_i are also modified as:

    $$\sigma_i^{\text{new}} = \sigma_i^{\text{old}} + 0.01 \cdot \sigma_i^{\text{old}} \cdot N_i(0,1).$$

  - All the σ_i are initialized usually by randomly sampling from $N(0,1)$ at the beginning of a run.

(cont'd)

- Now, we have the original μ points and their μ mutants. The fitness value of each of the 2μ points are compared to those of q points which are chosen randomly at every time of the comparison from the whole 2μ points. Then the 2μ points are ranked according to the number of wins, and the best μ points survive (q-tournament selection).

- The cycle of reconstructing the new population with better fitness points and restarting the search is repeated until one of the global optima is found or a set maximum number of generation has been reached.

• Evolution Strategy (ES)

$\star$ Adaptive-Mutation + Crossover = Evolution Strategy

(μ, λ) selection

- k offspring are generated from each of μ parents
- then the best μ offspring out of $\lambda = k\mu$ are selected.

$(\mu + \lambda)$ selection

- the best μ individuals are selected from μ parents and λ offspring $\Rightarrow$ *Elitist Strategy.*

(cont'd)

- Genetic Programming (GP)
  - ⋆ Chromosomes are tree-structures rather than strings.

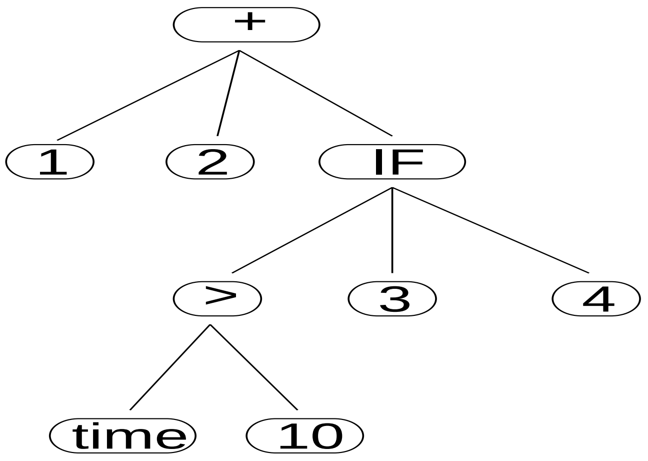
    If the trees are LISP S-expressions

    $\Downarrow$

    we might call GP *an evolution of programs.*

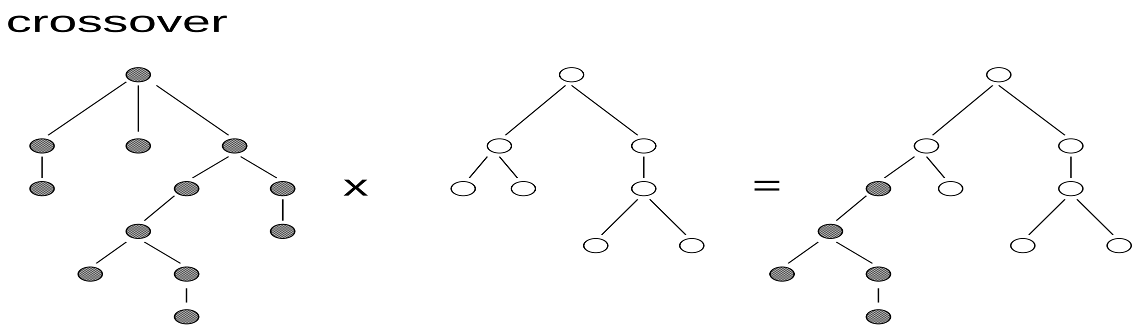
  - ⋆ An example of LISP S-expression:

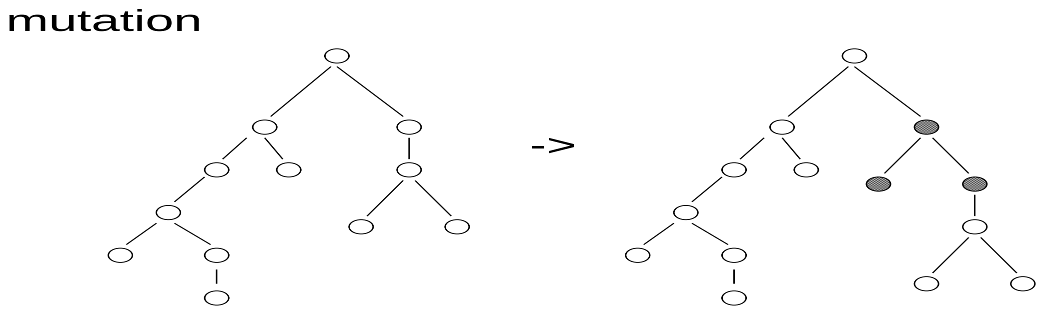
(+ 1 2 (IF (> time 10) 3 4))



(cont'd)

⋆ Crossover & Mutation in GP:





□ Commonly Used Test Functions:

— In order to learn more about EC's.

- Continuous Test Functions:

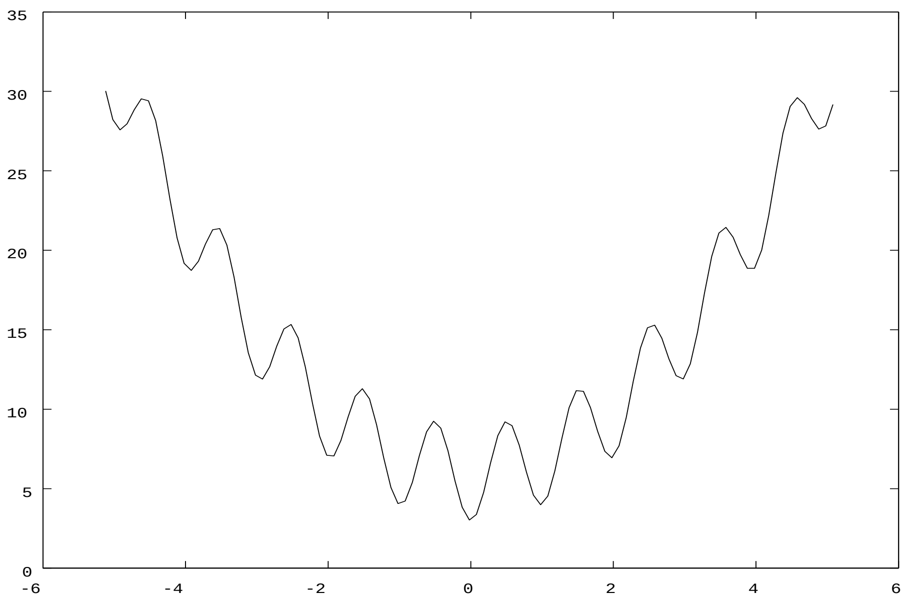
  ⋆ Rastrigin's Function F_6:

$$y = nA + \sum_{i=1}^{n} (x_i^2 - A\cos(2\pi x_i)), \quad x_i \in [-5.12 - 5.12].$$

  · Ruggedness might be controlled by modifying the value of A.

  · A two dimensional example ($n = 1$):

$$y = 3 + x^2 - 3\cos(2\pi x).$$



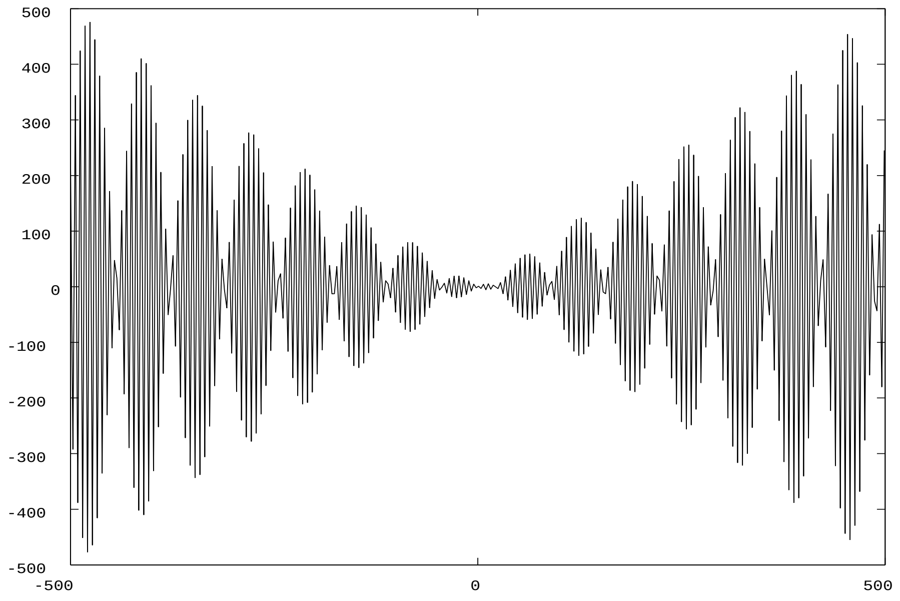
(cont'd)

⋆ Schwefel's Function F_7:

$$y = \sum_{i=1}^{n} (x_i \sin(|x_i|)), \quad x_i \in [-500, 500].$$

· A two dimensional example ($n = 1$):

$$y = x \sin(|x|).$$



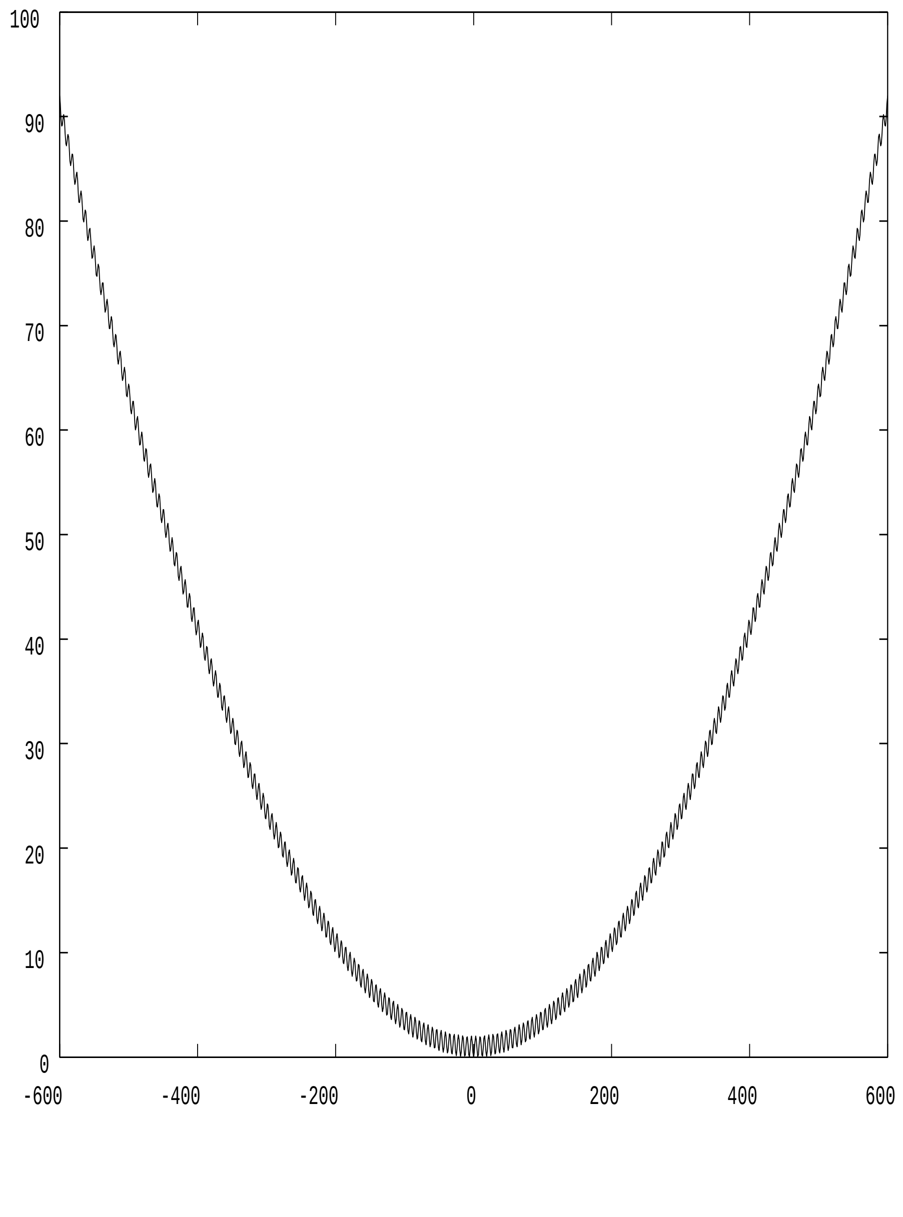
(cont'd)

⋆ Griewangk's Function F_8:

$$y = \sum_{i=1}^{n} x_i^2/4000 - \prod_{i=1}^{n} \cos(x_i/\sqrt{i}) + 1, \quad x_i \in [-600, 600].$$

· A two dimensional example:

$$y = x^2/4000 - \cos x + 1.$$



(cont'd)

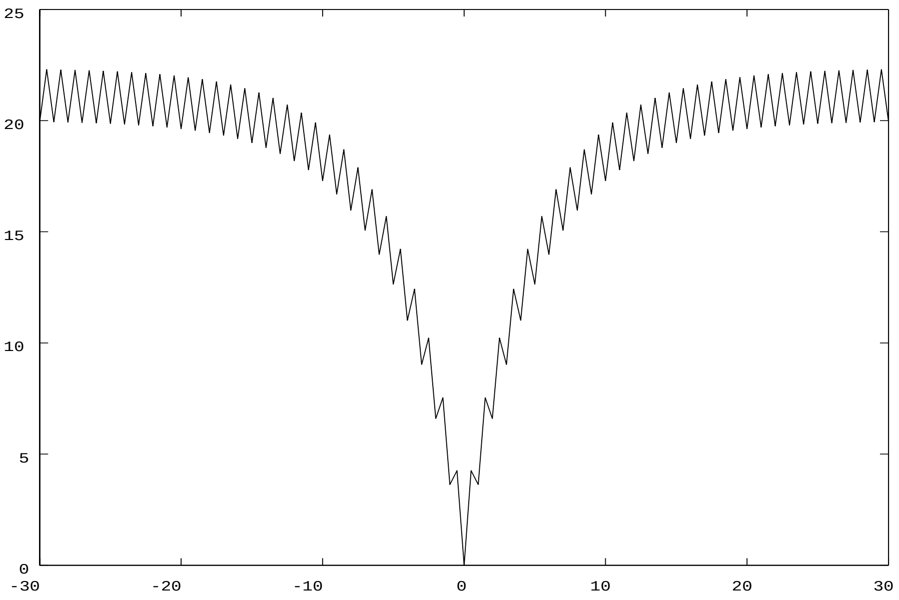
$\star$ Ackley's Function F_9:

$$y = -20 \sum_{i=1}^{n} \exp\left(-0.2\sqrt{x_i^2/n}\right) - \exp\left(\left(\sum_{i=1}^{n} \cos 2\pi x_i\right)/n\right) + 20 + e,$$

$$x_i \in [-30, 30].$$

· A two dimensional example:

$$y = -20\exp\left(-0.2\sqrt{x^2}\right) - \exp\left(\cos 2\pi x\right) + 20 + e.$$



(cont'd)

- De Jong's Five Test-Functions.

  $\star$ Sphere Model F_1:

$$\sum_{i=1}^{n} x_i^2, \quad x_i \in [-5.12, 5.12].$$

  $\star$ Rosenbrock's Function F_2:

$$\sum_{i=1}^{n-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)), \quad x_i \in [-5.12, 5.12].$$

  · This is not defined for $n = 1$ but only for $i \geq 2$.
  · A three dimensional example ($n = 2$)):

$$z = 100(y - x^2)^2 + x - 1.$$

  $\star$ Step Function F_3:

$$6n + \sum_{i=1}^{n} \lfloor x_i \rfloor, \quad x_i \in [-5.12, 5.12].$$

  · A two dimensional example ($n = 1$)):

$$y = 6 + \lfloor x \rfloor.$$

(cont'd)

⋆ Quartic Function with Noise F_4:

$$\sum_{i=1}^{n} i x_i^4 + N(0,1), \quad x_i \in [-1.28, 1.28].$$

· A two dimensional example $(n = 1)$):

$$y = 6 + \lfloor x \rfloor.$$

⋆ Shekel's Foxholes F_5:

$$\frac{1}{\dfrac{1}{500} + \dfrac{1}{\sum_{j=1}^{25} c_j + \sum_{i=1}^{2} (x_i - a_{ij})^6}}, \quad x_i \in [-1.28, 1.28].$$

where

$$(a_{ij}) = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & -16 & \cdots & -32 & -16 & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & -16 & \cdots & 32 & 32 & 32 & 32 & 32 \end{pmatrix}$$

(cont'd)

- NK Landscape
  — A fitness landscape where we can control its ruggedness.

⋆ Assume chromosomes with N genes each of which takes either 1 or 0.

- Each of its genes contributes to the fitness of the chromosome, depending, of course, on its allele value 1 or 0, but at the same time,
- the contribution is affected K other genes' 1 and 0.

  · The distribution of K among the N (nearest neighbors, at random, etc) is far less important than K and N.

$\Downarrow$ That is,

- The fitness contribution of the allele at the i-th gene depends on itself (1 or 0) and alleles at K other gene (total of K+1 alleles).

(cont'd)

⋆ How to assign fitness.

(1) Assign each gene i the K other genes which affect the gene.

(2) Assign a fitness contribution w_i to each gene in the context of K genes (which epistatically influence it) by drawing it randomly from uniform distribution between 0.0 and 1.0 to each of the 2^{K+1} combinations.

(3) Then the fitness of the chromosome is defined as

$$f = \sum_{i=1}^{N} w_i / N$$

Exercise 4 *Construct the following NK-landscape:*

(1) $N = 3$ *and* $K = 2$.

(2) $N = 4$ *and* $K = 2$.

(cont'd)

- Royal Road Function

  — To study the *Building Block Hypotheses.*

⋆ Royal Road Function $R(x)$ is a function to evaluate fitness of 64 bit binary chromosomes, i.e.,

- The fitness R(x) of a chromosome x is defined using a list of schemas s_i each of which is given with a coefficient c_i.

$$R(x) = \sum_{i=1}^{n} c_i \delta_i(x)$$

where

$$\delta_i = \begin{cases} 1 & \text{if } x \in s_i \\ 0 & \text{otherwise} \end{cases}$$

and s_i's are *schemata* specified as follows.

```
s1: 11111111################ ··· ########   c1 = 8
s2: ########11111111######## ··· ########   c2 = 8
s3: ################11111111 ··· ########   c3 = 8
                    .........
s8: ######################## ··· 11111111   c8 = 8
```

- The target is $(11111111 \cdots 11111111)$.

(cont'd)

⋆ The BBH suggests that GA works due to

(1) the presence of short, low-order, highly-fit schemas and

(2) the presence of intermediate stepping stones, that is, intermediate-order higher-fitness schema that result from combinations of the lower-order schemas and that, in turn, can combine to create even higher-fitness schemas.

And $R(x)$ explicitly contains these two features.

⋆ With BBH, one might expect that the building block structure will lay out a royal road for the GA to follow the string with higher order intermediate schema, like

11111111########**11111111**######## ··· ########

⋆ However, the performance is worse than a random search, e.g.,

Random Mutation Hill Climbing (RMHC) was 10 times better than the GA

⋆ Why? $\Rightarrow$ Hitch-hiking.

(cont'd)

— Random Mutation Hill Climbing (RMHC)—

(1) choose a string at random and call this current-hilltop

(2) choose a locus at random to flip. If the flip leads to an equal or higher fitness then set current-hilltop to the resulting string

(3) goto step (2) until an optimum string has been found or until a maximum number of evaluations have been performed.

(4) return the current-hilltop

(cont'd)

— Steepest Ascent Hill-climbing (SAHC) —

(1) choose a string at random and call this current-hilltop

(2) going from left to right, flip each bit recording the fitness

(3) if any of the resulting strings give a fitness increase then set current hilltop to the resulting string giving the highest fitness increase (ties are decided at random)

(4) if there is no fitness increase, then save current hilltop and goto (1), otherwise to (2) with the new current-hilltop.

— Next Ascent Hill-climbing (NAHC) —

(1) choose a string at random and call this current-hilltop

(2) same as SAHC except that as soon as a fitness increase is found go to step (2) without evaluating any more bit flips with the new current-hilltop and with the bit position being the one where the previous fitness increase was found

(3) if no such increases goto (1)

(cont'd)

- Then what is hitch-hiking?
  - ⋆ *Zeros hitch-hike a schema!*
    - Once an instance of a higher order schema is discovered, the schema spreads quickly in the population due to its high fitness.
    - with not preferable zeros in "#" positions along with the ones in the schema's defined positions.
  - ⋆ This slows the discovery of schemas which have ones in the other positions, especially those that are close to the highly fit schema's defined positions.
  - ⋆ The most likely positions for hitchhikers are those close to the highly fit schema's defined positions, since they are less likely to be separated for the schema's defined position under crossover.

□ Multi-modal Optimization.

— To obtain multiple solutions at a time in a run.

- Evolutionary methods to create and maintain multiple *species* in a population.
- These species independently search for a peak (hopefully an optimum solution), construct their *niche* and stay around the peak during a run.
- So-far proposed methods:
  - ⋆ Niching Method.
    - - Fitness sharing. [9]
      - · Similar individuals share fitness with each other.
    - - Crowding. [10]
      - · Similar individuals are replaced with random individuals
  - ⋆ Species Method.
    - · Mating is restricted to among similar individuals.

[9] Goldberg & Richardson (1987).
[10] De Jong (1975).

(cont'd)

- Each method more in detail.
  - ⋆ Fitness Sharing.
    - · Fitness of each individual is derated by an amount related to the number of similar individuals in the population.
    - · That is, shared fitness $F_s(i)$ of the individual i is

$$F_s(i) = \frac{F(i)}{\sum\limits_{j=1}^{\mu} s(d_{ij})}$$

where

- $F(i)$ is fitness of individual i;
- d_{ij} is distance between individual i and j;

  Typically d_{ij} is

$$\begin{cases} \text{Hamming distance} & \text{if in } \textit{genotypic space} \\ \text{Euclidean distance} & \text{if in } \textit{phenotypic space} \end{cases}$$

- $s(\cdot)$ is called *sharing function* and defined as:

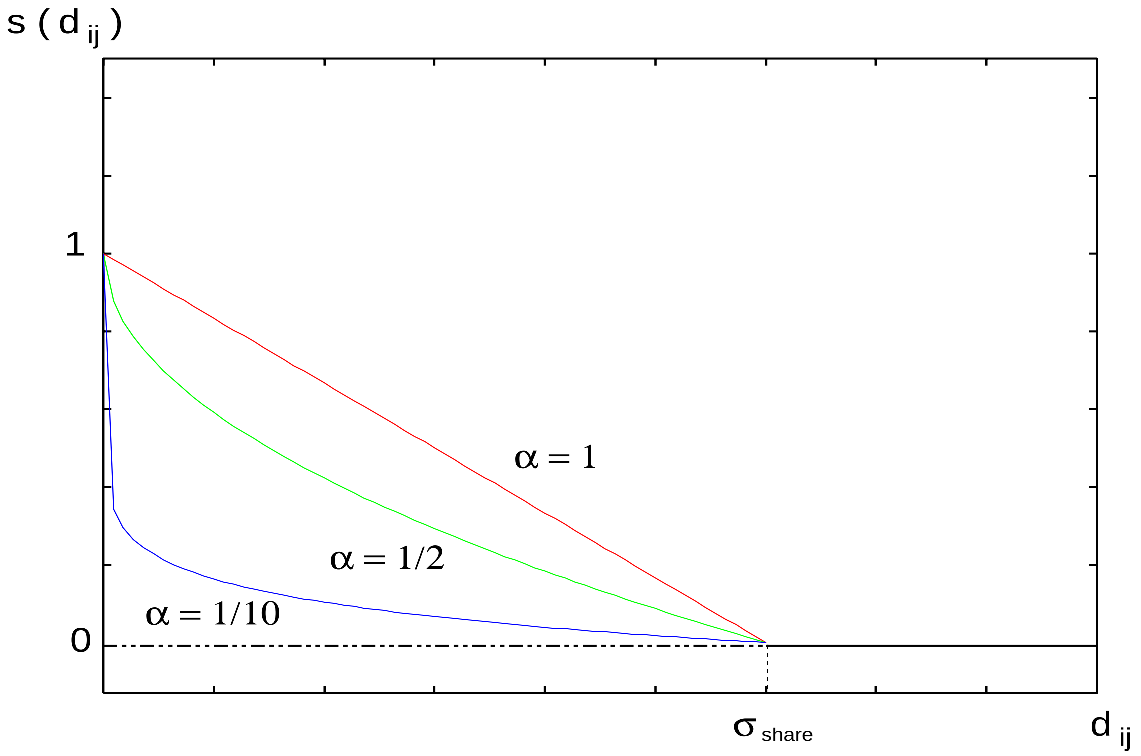
$$s(d_{ij}) = \begin{cases} 1 - (d_{ij}/\sigma_{\text{share}})^{\alpha} & \text{if } d_{ij} < \sigma_{\text{share}} \\ 0 & \text{otherwise} \end{cases}$$

  where

  - σ_{share} is interpreted as size of niche.
  - α determines the shape of the function.
- The denominator is called *niche count*.

(cont'd)

- Shape dependency of $s(d_{ij})$ on α



- To be short: Similar individual should share fitness.

$$\Downarrow$$

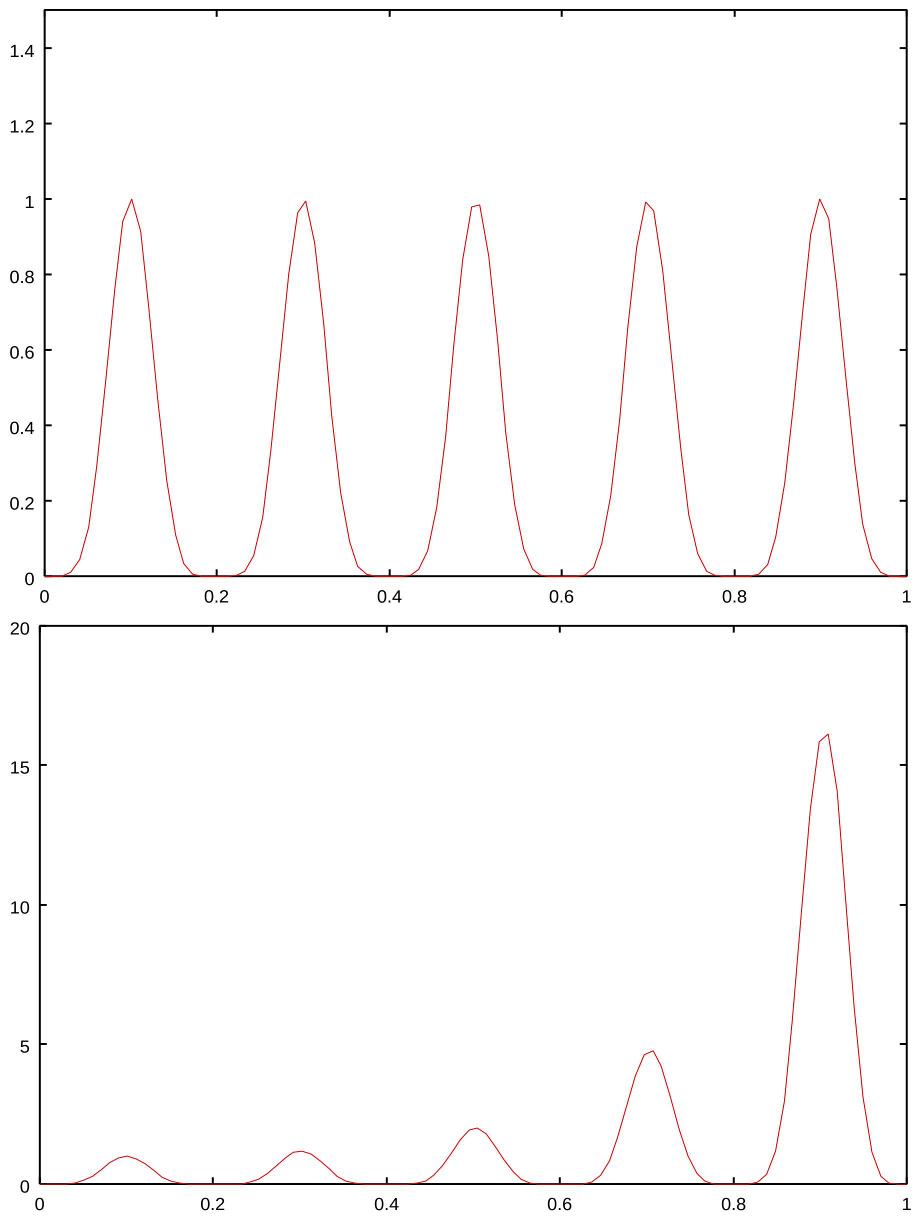
The number of individuals that can stay around any one of peaks (niche) is limited.

- The number of individuals stay near any peak will theoretically be proportional to the *hight* of the peak

(cont'd)

$$y = \sin^6(5\pi x)$$

$$y = -2((x - 0.2)/0.8)^2 \sin^6(5\pi x)$$



(cont'd)

— Gray Code —

- usual *(power-of-two)* binary code has Hamming cliffs, i.e.
  - two adjacent values may be many bits apart.
    - $15_{\text{decimal}} = 0111_{\text{binary}}$
    - $16_{\text{decimal}} = 1000_{\text{binary}}$
  - therefore Gray Code performs usually better in ECs
    - $15_{\text{decimal}} = 0100_{\text{gray-code}}$
    - $16_{\text{decimal}} = 1100_{\text{gray-code}}$
- how to create gray-code $a_1 a_2 \cdots a_n$
  from binary number$b_1 b_2 \cdots b_n$

$$a_i = \begin{cases} b_i & \text{if} \quad i = 1 \\ b_{i-1} \oplus p_i & \text{otherwise} \end{cases}$$

where $\oplus$ is addition modulo 2.

(cont'd)

⋆ Crowding.

- New random individuals are inserted into the population by replacing similar individuals.

- Similarity is determined by distance between individuals.
  - Here again distance refers to

$$\begin{cases} \text{Hamming distance} & \text{if in } \textit{genotypic space} \\ \text{Euclidean distance} & \text{if in } \textit{phenotypic space} \end{cases}$$

- The number of individuals gathering around a peak is largely determined by the size of each peak's basin of attraction under crossover (instead of hight of peaks.)

- In order to minimize the number of replacement errors Deterministic Crowding was proposed. [11]

[11] Mahfoud (1992).

(cont'd)

— Deterministic Crowding —

(1) Choose two parents, p_1 and p_2, at random, with no parent being chosen more than once.

(2) Produce two children, c_1 and c_2, with uniform crossover.

(3) Mutate the children by flipping bit chosen at random with probability p_m, yielding c_1' and c_2'.

(4) Replace parent with child as follows:

```
- IF  $d(p_1, c_1') + d(p_2, c_2') > d(p_1, c_2') + d(p_2, c_1')$
    * IF  $f(c_1') > f(p_1)$ THEN  replace $p_1$ with $c_1'$
    * IF  $f(c_2') > f(p_2)$ THEN  replace $p_2$ with $c_2'$
- ELSE
    * IF  $f(c_2') > f(p_1)$ THEN  replace $p_1$ with $c_2'$
    * IF  $f(c_1') > f(p_2)$ THEN  replace $p_2$ with $c_1'$
```

where $d(\zeta_1, \zeta_2)$ is the Hamming distance between two points (ζ_1, ζ_2) in pattern configuration space. The process of producing child is repeated until all the population have taken part in the process. Then the cycle of reconstructing a new population and restarting the search is repeated until all the global optima are found or a set maximum number of generation has been reached.

(cont'd)

⋆ Species Method.

· Drawbacks of the niching method.

- Effort of Niching method is to distribute and maintain individuals around each of the multiple optima.

  $\Downarrow$ Thus

  niching method cannot focus search on each optimum and as such, cannot find the exact location of the optima efficiently.

  $\Downarrow$ e.g.

  some of the search effort is wasted in the recombination of inter optima, which may produce individuals not close to any of the optimum.

  $\Downarrow$ So,

- Speciation restricts matings only with individuals near the optima, and

- discourages mating among individuals of different niches $\Rightarrow$ Both of the parents should belong to the same niche.
- This reduces the creation of lethal solutions (non of the optima) and make search efficient.

  $\Downarrow$ Therefor

- individuals belong to each peak should be known in advance.

  $\Downarrow$ Thus

· speciation cannot be used independently.

$\Downarrow$

"sharing + speciation" usually better performs than *"sharing alone"*.

· When individual i searches for a mate:

(1) Select individual j at random.

(2) If $r_{ij} < \sigma_{\text{mating}}$ then recombine i,

(3) otherwise search for another individual at random.

(4) Repeat (2) – (3) until suitable mate is found, or all population are exhausted, In this latter case pick up individual at random neglecting r_{ij}.

□ Multi-objective Optimization.

Sometimes in real world problems
multiple objectives should be optimized simultaneously.

$\Downarrow$

There may be no one solution that is best w.r.t. all objectives

or

some objectives conflict:

$\Downarrow$

e.g. performance, reliability, and cost

(cont'd)

- Parete optimal (non-dominated) solutions:

Suppose we have n objectives (n fitness functions)

$$f_1(\mathbf{x}), f_2(\mathbf{x}), \cdots, f_n(\mathbf{x})$$

where $\mathbf{x}$ is a candidate solution.

$$\Downarrow$$

If a new solution $\mathbf{y}$ improves all the objectives for $\mathbf{x}$, i.e.

$$f_i(\mathbf{y}) > f_i(\mathbf{x}) \quad \text{for} \quad \forall i$$

the new solution $\mathbf{y}$ is said to *dominate* the old solution $\mathbf{x}$.

$$\Downarrow$$

If it's impossible any more (no such $\mathbf{y}$ exists), then $\mathbf{x}$ is referred to
non-dominated or *Parete optimum.* [12]

[12] Sometimes, not all Pareto optimal solutions are acceptably compromise. It sometimes a matter of human preference.

(cont'd)

- A toy example

$$\begin{cases} f_1(x) = x^2 \\ f_2(x) = (x-2)^2 \end{cases}$$

· x=0 is optimum w.r.t. f_1 but not so good w.r.t. f_2.

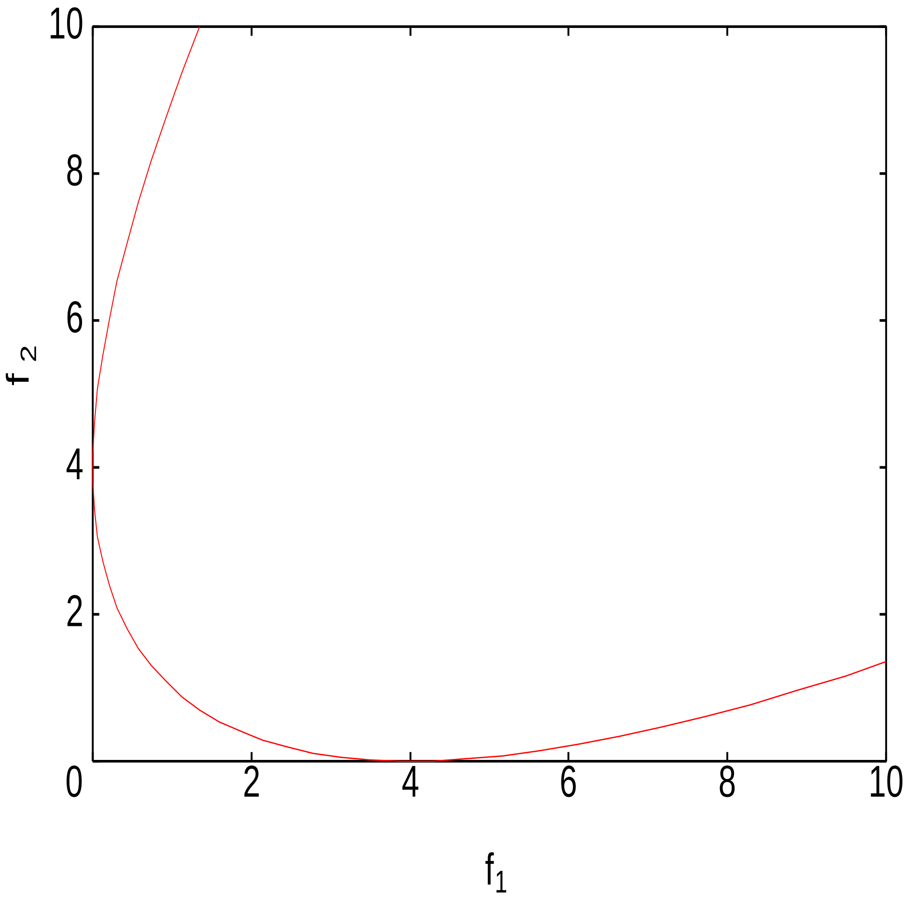
· x=2 is optimum w.r.t. f_2 but not so good w.r.t. f_1.

· Any other point in between is a compromise or trade-off and is a Pareto-optimum.

· But the solution x=3, eg, is not a Pareto-optimum since this point is not better than the solution $x = 2$ w.r.t. either objective.

(cont'd)

- Trade-off Space



Thought Experiment 1 *What if we plot all individuals of generation 0 and, say, generation 100?*

(cont'd)

In real world problems we often need different alternatives in decision making and want multiple Pareto-optimal solutions simultaneously.

$$\Downarrow$$

- Two methods (out of many)
  - ⋆ Weighted-sum approach: [13]

$$f(\mathbf{a}_i) = \sum_{k=1}^{n} w_k \cdot f_k(\mathbf{a}_i)$$

    - · For any set of positive weights, the weighted sum of n fitness functions is guaranteed to be Parete optimality. [14] (But opposit is not true.)

  - ⋆ Exploration of non-dominated region. [15]
    - · Each individual is assigned a fitness value according to how many other individuals in the population dominant the individual.
    - · Then good ones survive after tournament selection.

[13] Nondominated solutions located in concave regions of the trade-off surface cannot be obtained in this method.
[14] See Goicoechea et al. (1982).
[15] Fonseca and Flemming (1993).

□ When to Compare an Algorithm to Others.

Theorem 2 (No Free Lunch Theorem)
When applied across all possible problems, all algorithms that do not resample points from the search space perform exactly the same on average (Wolpert & Macready, 1996).

$\Downarrow$ one of implications

It is necessary for search algorithms to incorporate
some knowledge of the search space
in order to perform better than random search.

□ Summery and Conclusions.

- Forget theory for a time being.

- Just start and try to apply EC to your problems.

  It's easier than you imagine

□ Related Web-pages

- **Hitch-Hiker's Guide to EC**

  http://surf.de.uu.net/encore/www

- **The Genetic Algorithms Archive**

  http://www.aic.nrl.navy.mil/galist

- **Illinois Genetic Algorithms Laboratory (IlliGAL)**

  http://www-illigal.ge.uiuc.edu

- **Genetic Programming Notebook**

  http://www.geneticprogramming.com

- **Alife Online**

  http://alife.org